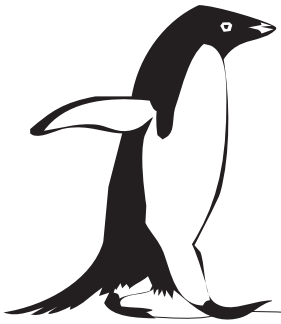


Linux大棚命令百篇（下）

网络和系统篇 GO

吴鹏冲 杨文强 张昱 编著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书打破了市面上主流 Linux 命令书籍的写作风格，创新性地以专题文章或系列文章的形式来组织全书，文风轻松通顺、循序渐进，既适合作为系统学习的案头书，也适合在床头边、地铁上、院落中阅读。

本书是这套系列丛书的第二本，内容侧重在网络和系统方面。为了体现知识的结构化、系统化，本书共分为三篇。

第一篇 网络篇

这一部分是本书的重中之重，囊括了 Linux 工程师最常用的网络相关命令，通过对本篇的学习，读者将全面掌握 Linux 系统网络层面的各类知识和技能，包括用于网络测速的 ping 命令、用于域名解析的 nslookup 命令和 dig 命令、用于网络配置的 iproute2 套装、用于流量分析的 tcpdump 工具、用于建立系统信任关系的 ssh-copy-id 命令、用于数据网络同步的 rsync 工具，以及用于网络数据下载的 wget 命令，等等。

第二篇 进程和性能篇

这一部分专注于系统进程、服务器资源和性能方面。作为一名 Linux 工程师，总是希望能够全面了解服务器资源使用情况，快速定位系统性能瓶颈，那么，阅读和学习这一篇将是最好的选择。本篇将告诉大家 free 命令的很多不为人知的学问、SWAP 的进阶知识、多核 CPU 的查看方法、top 命令的使用技巧、vmstat 输出内容中的指标含义、kill 命令如何精准地杀死进程，等等。

第三篇 系统管理篇

这一部分专注在系统管理方面，主要介绍了和 Linux 操作系统原理相关的知识，包括查看系统基本信息的 uname 命令、查看用户账户的 who 命令、控制服务等级的 chkconfig 命令、查看机器硬件配置的 dmidecode 命令，等等。

学习完本书后，相信读者朋友们可以轻松而愉快地掌握 Linux 的网络、系统性能、系统管理等知识和技能，并达到一线互联网公司 Linux 工程师的水平。

图书在版编目（CIP）数据

Linux 大棚命令百篇. 下，网络和系统篇 / 吴鹏冲，杨文强，张昱编著. —北京：电子工业出版社，2016.7
ISBN 978-7-121-29371-9

Ⅰ. ①L… Ⅱ. ①吴… ②杨… ③张… Ⅲ. ①Linux 操作系统－程序设计 Ⅳ. ①TP316.89

中国版本图书馆 CIP 数据核字(2016)第 159476 号

责任编辑：安　娜
印　　刷：
装　　订：
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：787×980　1/16　印张：18　字数：345 千字
版　　次：2016 年 7 月第 1 版
印　　次：2016 年 7 月第 1 次印刷
印　　数：3000 册　　定价：59.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819　faq@phei.com.cn。

推荐序 1

日月如梭，自 1991 年 10 月 Linus Torvalds 发布第一版 Linux 操作系统以来，经过 25 年的历程，这个基于自由和开放源代码模式的操作系统已经日益发展壮大。从嵌入式系统、智能手机和平板电脑、个人计算机、网络服务器、云计算到高性能超级计算系统，到处可以看到它的身影。据 Gartner 等国际机构的统计，作为操作系统的重要成员，Linux 在各类操作系统中所占的比重越来越大。

人们说 Linux 是个类似 UNIX 的多用户、多任务操作系统，是说 Linux 继承了很多 UNIX 的优秀特性，具备了模块化的设计，其进程控制、文件系统、外部设备、网络功能、安全管理以及各种功能齐全和强大的工具软件，可以方便地控制计算机系统完成各种操作，具备了免费和开源特性的 Linux 操作系统随着互联网在各个领域的发展，得到了更加快速的普及应用。从 1996 年起就支持 IPv6 协议的 Linux 对推进下一代互联网的部署发挥了重要作用。

Linux 操作系统得到迅猛的发展，这与 Linux 具有的良好特性是分不开的，包括免费和开放特性、多用户多任务处理能力、方便灵活且功能强大的的 Shell 命令、丰富灵活的多种网络通信命令、可靠的系统安全措施、对多种多样外部设备的支持，以及良好的可移植性。

要想使用好 Linux 操作系统，充分发挥它的能力，就要学习好 Linux 的使用方法。现有关于 Linux 的书籍已经出版了很多，但这本书是非常有特色的一本。作者运用十分幽默风趣的语言，从 Shell 命令开始，介绍了文件编辑与内容处理，文件的查找、压缩与硬盘管理，网络相关命令，进程与性能调优，Linux 系统管理等各种命令的使用方法和技巧。

无论是初学者学习使用 Linux，还是开发者或系统管理员作为常用工具手册，这本书都是十分值得拥有的。一本好的入门教材会让初学者快速领悟到 Linux 系统的基

本使用方法，掌握常用的 Linux 操作命令。如果仅仅依靠系统自带的 man 命令，往往会令初学者感到云遮雾罩，不明所以。而对 Linux 系统管理员来讲，本书对网络命令、系统调优等命令的介绍，可以使你对这些命令及其显示结果有更深入的理解。书中还列举了很多 Linux 发展历史中的趣味小故事，使读者在掌握 Linux 使用方法的同时，也调节了心情，增加了乐趣。

正像篇首所说，日月如梭，Linux 已经面世二十五年啦。本书的作者从进入我们研究室学习到毕业工作，也已经十年了。应作者的邀请，作为本书的首批读者，我怀着兴奋的心情一边浏览着各个篇章，一边回忆着这些年来互联网的发展，以及他们的成长历程。他们有多年的工程实践经验，在大型网络公司掌管着上千台 Linux 集群服务器的运行与维护工作，积累了丰富的 Linux 使用经验和技巧。我诚挚推荐读者来阅读本书，也期待着他们能为读者带来更多的新作。

马严

北京邮电大学网络技术研究院教授、博士生导师

推荐序 2

技术，一直是驱动社会不断进步和发展的主要动力。从蒸汽时代、电力时代到今天的信息时代，技术始终是推进社会发展的第一生产力。放眼未来，互联网+正推动互联网与社会各行业深度融合，人工智能、云计算、物联网、自动驾驶技术蓬勃发展，人类正在经历着第四次全球性科技革命。而我们有幸身处其中，掌握新时代核心技术的人才已经成为这轮洪洪巨流的推动者。

Linux，自从 1991 年发布至今，对计算机技术，互联网行业产生了巨大的推动作用。互联网时代，Linux 无处不在，占据了全球绝大部分的服务器份额。这与 Linux 操作系统本身的高度开放性、高可定制性、高可用性等是密不可分的。百度等众多中国互联网企业的技术体系都是基于 Linux 操作系统构建的，熟练掌握并精通 Linux 技术，是互联网技术从业者的必备技能和核心竞争力之一。

无论是在校学生还是已入职场的工程师，学习并掌握 Linux 系统技术，需要一个边学习边实践的过程，并在解决实际问题中融会贯通。在国内互联网技术发展的早期，Linux 优质资料稀缺、应用场景匮乏，国内工程师只能借鉴国外资料，学习梯度极高，全行业严重缺少高水平的系统管理人才，与国外同业差距明显。时至今日，中国互联网的蓬勃发展领先全球，国内也逐渐培养出一批具备先进实战经验的 Linux 系统人才，他们或掌管着中国互联网的基础设施，或运营着大规模集群，或构建出复杂的系统架构，或已经成为行业级系统架构师等领军人物。国内完全有条件诞生一部既有 Linux 基础又有经典实践经验的优秀著作，帮助读者快速地汲取经验，成为专家。

鹏冲曾在百度运维部磨练七年，先后担任垂直搜索运维团队技术负责人，全百度统一监控平台产品负责人等重要岗位，在 Linux 系统和集群管理方面拥有着深厚的技术积累和实践经验。这套关于 Linux 命令进阶的丛书是他多年积累的经验输出。

我有幸比广大读者更早阅读了本书，整个阅读体验顺畅，对于 Linux 常用命令的讲解力求深入浅出，并将实际应用中需要掌握的技术点讲解得相当透彻。对于从事或有志于从事互联网技术工作的读者，这本书将帮助大家从实用的角度学习和积累。

我推荐各位 Linux 技术从业者阅读和学习，相信这会是一个正确的选择。

李硕

百度运维部总监

自序

北邮七年学习，百度七年工作，让我经历了很多，思考了很多，也收获了很多。

知乎是我很喜欢的一个问答社区，“×××是一种怎样的体验？”“如何评价×××？”早已成为时下最流行的提问姿势。

所以呢，我会尝试着模仿知乎的提问风格，和大家分享我的五点思考和体会：

1．这本书为什么值得读？

2．为什么建议大家写博客？

3．如何进行知识管理？

4．如何学好 Linux？

5．在百度运维部工作是一种怎样的体验？

【这本书为什么值得读？】

虽然有种老王卖瓜的感觉，但我还是鼓起勇气，希望能用三个足够客观的理由吸引到你。

（1）聚焦专题：以专题和系列文章的形式来讲解知识，是本书的一大特点。读者可以在一段较短的时间内，聚焦在一个命令的学习上，集中精力实现进阶。

（2）贴近实战：书中内容全部来自于作者长期从事大规模 Linux 集群运维的经验总结，确保了本书的实用性。通过阅读本书，读者的 Linux 命令掌握水平可以更快地达到一线互联网公司 Linux 工程师的水平。

（3）易于阅读：作者长期在“Linux 大棚”从事技术博文的写作，善于用简单的语言、清晰的文章结构来解释复杂晦涩的概念和知识，让用户可以非常顺畅地阅读和理解。

【为什么建议大家写博客？】

我在 2008 年 9 月创立了 Linux 大棚博客，一直坚持写作至今。我和大家分享写作的四点好处：

第一，觉得懂未必懂。写作是自我反省、自我提升的一个过程。不把知识落成文字，你就不会发现你掌握着许多模棱两可和模糊不清的知识。

第二，让别人懂才是真的懂。写作正是在强迫你给别人讲懂知识。在写作过程中，你需要思考应该先讲哪些知识，后讲哪些知识，需要思考应该通过哪些场景引出哪些知识，需要思考应该如何做知识的类比。这些技巧看似容易，实则并不容易。

第三，看似浪费时间，实则节省时间。知识总会被遗忘，但有实验证明阅读自己写过的知识，可以更快地重新掌握。所以，为了节省时间，请多写作。

第四，交到朋友还能出书。通过博客写作，可以吸引到不少志同道合的朋友，可以和他们一起交流一起进步。如果文章内容还不错，说不准会有出版社的编辑联系你出书哦。

【如何进行知识管理？】

每个人都有自己的一套知识管理的方法，而我只是抛砖引玉。

按照知识的规模分，我将知识分成三种类型：

（1）小型知识：往往是一句话或一个段落就能说清的知识，如一个技术名词的解释、一个命令的使用技巧等。

（2）中型知识：需要一篇文章，甚至一个系列的文章才能介绍清楚的知识，如一个命令的完整用法、几种数据库技术的比对和选型等。

（3）大型知识：需要一本书或多本书才能讲解清楚的知识，如 Linux 系统、MySQL 数据库技术等。

按照知识的公开度分，我把知识分成两类：

（1）愿意公开的：比如一些公共知识，不含个人信息，也不含保密信息的。

（2）不愿意公开的：比如一些含有保密信息的知识，一些自己的随笔等。

而基于这两种分类方法，我一般会采用不同的手段，管理不同的知识：

（1）小型知识、愿意公开：微博（比如“Linux 大棚”官方微博）；

（2）中型知识、愿意公开：博客（比如“Linux 大棚”技术博客）；

（3）大型知识、愿意公开：书籍（比如这本书）；

（4）小型知识、不愿公开：云笔记；

（5）中大型知识、不愿公开：本地 World 文档、自建私有 Wiki。

你会发现大部分的知识，都可以对应到上面的分类中。

当然，知识管理和减肥是一个道理，知易行难，一定要坚持养成知识管理的习惯，长此以往，才能受益。

【如何学好 Linux？】

从我的个人学习经历来看，“系统学习+实践+写作+交流分享”是学习 Linux 技术的一套有效的组合拳。

系统学习，即通过优秀的书籍、培训视频、培训课程等方式来系统地学习 Linux 系统。

实践，即真正到 Linux 环境中去学习，去工作，去主动解决问题。我在学习 Linux 之初，就在笔记本中完整安装了 Fedora 系统、Ubuntu 系统、Debian 系统和 FreeBSD 系统，来强迫自己在 Linux 环境中办公和娱乐。

写作，就是要养成写文章的习惯，把自己觉得模糊的知识点写成可发表的文章，这时候，你会发现，很多细节知识，你都要反复思考和查证，这个过程，就是进阶的过程。

交流分享，建议去结识一些 Linux 技术的高手和专家，他们的一些经验和体会，或许能让你事半功倍。

【在百度运维部工作是一种怎样的体验？】

据我所掌握的信息来看，百度运维部应该是国内承担着超大规模 Linux 服务器运维任务的少数团队之一，Linux 服务器规模达数十万。

由于规模效应的影响，在这里工作，即便是发生概率为 0.1%%的 BUG，都可能会每天发生。所以，在这里工作的运维工程师要面临的问题和挑战，将是国内同行很少碰到的，当然，据此而积累的经验和锻炼的解决问题的能力，也是国内顶尖的。

在百度的技术体系中，运维部处于研发部和系统部之间，研发部负责百度产品的

开发工作，系统部负责操作系统、服务器、网络、机房等设施，而运维部则负责操作系统及上面运行的服务，确保服务的高可用性，同时不断地提升效率，降低成本。

就拿我曾负责的百度视频产品运维来说，运维工程师首先要确保的是服务的可用性，也就是要确保全国网民都可以访问到百度视频服务；其次，要通过 CDN、缓存等多种技术手段不断提升网民访问网站的速度，提升网站访问体验；再者，需要更准确地监控到线上故障，更快速地实现模块升级、更可靠地实现故障自动化处理；最后，就是要追求更少的机器成本、更低的带宽成本、更少的人力投入来实现同样质量的运维服务。

有人会说做运维工作很辛苦，其实我想说，作为七年运维人，我一直相信，运维是架构师的必备技能之一，不具备运维经验和视野的人，是很难设计出优秀的架构的。不经一番寒彻骨，怎得梅花扑鼻香。

这篇自序，包含了几个方面的信息，都是我希望和大家分享的，也相信是大家所希望了解的。好了，如果大家对其中的哪些内容感兴趣，欢迎与我联系，我们深入沟通。下面的时间，就交给大家，来好好阅读这本书吧！

目录

网络篇

在网络篇中，我们将为大家带来 29 篇文章，所有内容都是围绕着网络技术和工具展开的，具体如下：

在这里，你不仅可以用 nslookup 和 dig 工具玩转 DNS，还可以与神探 tcpdump 一起去网络流量中探险，还可以了解到 netstat 被 ss 篡权的幕后故事，想想都激动！

让我们现在就开始网络篇的学习之旅吧。

1 ping 遍大江南北

ping 不止是 ping

接触过计算机网络的同学一定都知道 ping 命令吧，当计算机联网出现问题时，第一个进入脑海的解决方法就是“ping 一下网络呗”。ping 是如此的常用，它绝对是你使用频率最高的一条网络命令了。

在 Windows 系统中，我们更多的是简单地 ping 一下网站，来测试网络连通性，就像这样：

```
ping roclinux.cn
```

但作为更专业的 Linuxer，知道这些还远远不够，今天我们就为大家更全面地介绍一下这位最熟悉的陌生人——ping 命令。

指定 ping 的次数

受 Windows 使用习惯的影响，在 Linux 系统中需要 ping 的时候，你也许会这样：

```
[roc@roclinux ~]$ ping roclinux.cn
PING roclinux.cn (116.255.245.206) 56(84) bytes of data.
64 bytes from 116.255.245.206: icmp_seq=1 ttl=49 time=16.3 ms
64 bytes from 116.255.245.206: icmp_seq=2 ttl=49 time=16.7 ms
64 bytes from 116.255.245.206: icmp_seq=3 ttl=49 time=15.9 ms
64 bytes from 116.255.245.206: icmp_seq=4 ttl=49 time=19.1 ms
64 bytes from 116.255.245.206: icmp_seq=5 ttl=49 time=18.1 ms
```

当你满怀期待的等着命令自己结束，可等到花儿都谢了，命令还是在执行。这是怎么回事呢？原来，Windows 下的 ping 命令和 Linux 下的是有所不同的，Linux 下的 ping 必须指定次数，不然它会无限次地执行下去。

如何指定执行次数呢？很简单，只需使用-c 选项来设定次数即可。比如，ping 三次 roclinux.cn 网站的正确执行方法是：

```
[roc@roclinux ~]$ ping -c 3 roclinux.cn
PING roclinux.cn (116.255.245.206) 56(84) bytes of data.
64 bytes from 116.255.245.206: icmp_seq=1 ttl=49 time=15.9 ms
64 bytes from 116.255.245.206: icmp_seq=2 ttl=49 time=16.4 ms
64 bytes from 116.255.245.206: icmp_seq=3 ttl=49 time=15.7 ms

--- roclinux.cn ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 15673ms
rtt min/avg/max/mdev = 15.785/16.084/16.496/0.301 ms
```

只重结果不重过程

不想看到 ping 命令那啰唆的过程，只想看到结果统计，因为只有结果才是最关键的。那有没有办法满足大家这个小小的愿望呢？好消息!强大的ping 命令说 so easy! 只需简单地增加-q 选项就行了。不信的话，试试下面的命令：

```
[roc@roclinux ~]$ ping -q -c 3 roclinux.cn
PING roclinux.cn (116.255.245.206) 56(84) bytes of data.

--- roclinux.cn ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 16794ms
rtt min/avg/max/mdev = 15.869/15.994/16.161/0.160 ms
```

这样的 ping 绝对是模范员工，工作时埋头苦干，出报告时清晰准确。

这时，我突然发现了一个陌生的指标 mdev，就在 ping 命令输出内容的最后一行，这个指标是用来干什么的呢？

原来 mdev 是 Mean Deviation 的缩写，表示 ICMP 包的 RTT 偏离平均值的程度，主要用来衡量网速的稳定性。mdev 的值越大说明网速越不稳定。

另外，不同的操作系统的 mdev 的名字也有所不同，在 mac 下它叫作 stddev，而在 Windows 下则根本没有这个统计指标。

指定 ping 的数据包的大小

默认情况下，ping 命令是以 64 字节大小的数据包来测试网络联通性的，如需要改变默认数据包的大小，则可以使用参数-s 选项。比如你想使用 65500 字节的数据包来测试网络，命令可以这样来写：

```
[roc@roclinux ~]$ ping -s 65500 -c 3 roclinux.cn
PING roclinux.cn (116.255.245.206) 65500(65528) bytes of data.

--- roclinux.cn ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 11999m
```

在实际工作中，我们通常使用-s 选项来发现网络环境中有关 MTU 的问题。

指定 ping 的 TTL

TTL，即生存时间，是指数据包被路由器丢弃之前允许通过的路由器跳数。

TTL 是由发送主机来设置的，为了防止数据包在网络中无限循环，每个路由器在转发网络数据包时，都要求将 TTL 的值减少 1，直到 TTL 减为 0 的那一刻，也就是这个数据包生命终结的时刻。

对于 ping 命令发出的数据包，我们可以通过选项，来设定它在网络上的生命时长：

```
ping -t 255 roclinux.cn
```

如果我们不使用-t 选项来设置 TTL，那么 ping 命令会采用 TTL 默认值。而不同的操作系统 TTL 默认值也是不相同的。

下面给大家列举一些常用操作系统的默认值。

- Linux 系统的 TTL 值为 64 或 255
- Windows NT/2000/XP 系统的 TTL 值为 128
- UNIX 系统的 TTL 值为 255

指定 ping 的时间间隔

ping 命令的核心功能就是查看网络的联通性和网络的延迟。默认情况下，发送两个数据包之间的间隔是 1 秒，如果我们嫌默认 1 秒发送一个 ping 包太慢，则可以使用-i 选项来指定发送两个 ping 包之间的时间间隔。不过需要注意的是，只有 root 用户才能设置低于 0.2 秒的时间间隔。下面我们就以 root 用户实现 0.1 秒时间间隔的 ping 命令：

```
[root@roclinux ~]# ping -i 0.1 -c 3 roclinux.cn
PING roclinux.cn (116.255.245.206) 56(84) bytes of data.
64 bytes from 116.255.245.206: icmp_seq=1 ttl=49 time=16.1 ms
64 bytes from 116.255.245.206: icmp_seq=2 ttl=49 time=16.6 ms
64 bytes from 116.255.245.206: icmp_seq=3 ttl=49 time=16.1 ms

--- roclinux.cn ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2119ms
rtt min/avg/max/mdev = 16.134/16.331/16.699/0.260 ms
```

爱如潮水般的 ping

上文我们看到，ping 命令在默认情况下使用 1 秒作为发送间隔，而使用-i 选项可以显式地指定发送间隔。如果我们希望 ping 命令以尽可能快的速度来发送数据包，则可以使用-f 选项来实现爱如潮水般的 ping：

```
 [root@roclinux ~]# ping -f -c 100 roclinux.cn
PING roclinux.cn (116.255.245.206) 56(84) bytes of data.

--- roclinux.cn ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 1473ms
rtt min/avg/max/mdev = 15.444/16.269/21.434/1.082 ms, pipe 2, ipg/ewma
14.888/15.739 ms
```

-f 选项，即 flood ping，潮水模式的 ping，听起来就无比威武，大有天降雄师的阵势。

这里有一个知识点要注意，flood ping 会采用无间隔的方式尽全力发送探测数据包，确保每秒钟至少发送 100 个。我们把这种模式形象地称为“疯狂模式”，注意，这种方式只有 root 用户才可以使用。

下面我们就使用 flood ping 来测试一下网卡的丢包率：

```
[root@roclinux ~]# ping -f -c 10000 192.168.0.29
PING 192.168.0.29 (192.168.0.29) 56(84) bytes of data.

--- 192.168.0.29 ping statistics ---
10000 packets transmitted, 10000 received, 0% packet loss, time 144ms
rtt min/avg/max/mdev = 0.003/0.004/0.043/0.003 ms, ipg/ewma 0.014/0.005
ms
```

“0% packet loss”表示丢包率为 0，说明网卡工作非常正常，也没有任何网络拥塞发生。如果你想查看一下你们公司内部的网络状况的话，只需将上述 IP 改成你们公司机器的 IP 地址就可以了。

篇尾小福利

最后，我们给出在网络实际使用过程中的一些 RTT 参考值，希望对你排查网络问题有一定的帮助，如表 1 所示。

表 1　RTT 参考值

场　　景	RTT 参考值
ping 本机	0.01ms
ping 同机房机器	0.1ms

续表

场　　景	RTT 参考值
ping 同城机器	1ms
ping 不同城机器	20ms
北（南）方 ping 南（北）方机器	50ms
从国内 ping 国外机器	200ms

提醒大家：网络很复杂，情况很多变，上述数据仅供参考。

好了，相信大家又重新认识了一下老朋友——ping，也学习了一些新玩法，希望能对大家的网络问题排查有所帮助。

2 DNS 探秘之一——nslookup 初体验

DNS 和 nslookup 的关系

DNS，即 Domain Name System，中文称之为“域名系统”，是计算机网络世界中非常重要的一个角色，负责着整个互联网中“域名—IP 地址”的管理和解析工作。

自从有了 DNS，互联网就变得友好了许多，人们访问网站时不必再去记忆那些晦涩的 IP 地址，通过一些非常易懂的字串就可以方便地指定目的网站。

我们平时都是通过 www.baidu.com 来访问百度的，很少有人会记住它的 IP 地址吧！如果你对大型网站的域名管理有所了解的话，你会知道 IP 地址解析还存在着就近解析和经常更换的问题，所以，记忆 IP 地址并不切实际。

简单介绍了 DNS 之后，我们抓紧时间来为大家介绍一下今天的主角——nslookup。

通过 man nslookup，我们可以看到 nslookup 的官方解释是“query Internet name servers interactively”。

而 nslookup 是 name server lookup 的缩写，顾名思义，nslookup 就是“用来查询 DNS 的”。假如你想知道 www.baidu.com 对应的 IP 地址的话，那么用 nslookup 应该是最正确的方法了。

系统没有 nslookup 命令怎么办

如果你的 Linux 系统中没有 nslookup 命令，那么八成是你的系统中没有安装 bind-utils 软件包。

bind-utils 软件包中包括了我们操作和管理 DNS 的一系列工具，比如 host 命令、dig 命令、nslookup 命令等等。

安装 bind-utils 软件包，并非难事，在 RHCE、CentOS 或 Fedora 上，通过一条命令就可以搞定了：

```
yum install bind-utils
```

nslookup 的两种模式

前面我们提到 nslookup 的官方解释是“query Internet name servers interactively”，这里的 interactively 说明 nslookup 是具有交互功能的。

的确，nslookup 共有两种工作模式，一种是“交互模式”，另一种则是“非交互模式”。

- 在“交互模式”下，用户只需执行一次 nslookup，就可以向域名服务器连续发起查询请求。
- 在“非交互模式”下，用户发起的查询请求是一次性的，下次再想查询，就需要再执行一次 nslookup。

如何进入交互模式

nslookup 的交互模式使我们可以连续发起 DNS 查询请求，而不必每次都运行 nslookup 命令。

进入交互模式的方法也很简单，只需输入 nslookup 命令，无须加任何参数，就可以直接进入交互模式啦！

```
#进入到了nslookup的交互模式
[roc@roclinux ~]$ nslookup
>
```

看到最后的右尖括号“>”了吧，这就是 nslookup 进入交互模式的重要标志。

需要注意的是，此时 nslookup 会连接到默认的域名服务器，也就是/etc/resolv.conf 中所配置的第一个 DNS 服务器地址。

如果我们想自己来指定一个 DNS 服务器地址，也是完全可以的，只需设置 nslookup 的第一个参数为“-”，而第二个参数是要连接的 DNS 服务器 IP 地址即可。

假如我们想连接到谷歌的开放 DNS，则可以这样来做：

```
#我们要连接到Google的开放DNS服务器
[roc@roclinux ~]$ nslookup - 8.8.8.8
>
```

下面就在交互模式下实战一次，来查询 www.baidu.com 域名的 IP 信息：

```
[roc@roclinux ~]$ nslookup - www.baidu.com
> www.baidu.com
Server:        8.8.8.8
Address:       8.8.8.8#53

Non-authoritative answer:
www.baidu.com   canonical name = www.a.shifen.com.
Name:   www.a.shifen.com
Address: 61.135.169.125
Name:   www.a.shifen.com
Address: 61.135.169.121
>
```

可以看到，Google 的开放 DNS 服务器给我们返回了两个可用的 IP 地址，即 61.135.169.125 和 61.135.169.121。而且在这次查询请求之后，提示符仍然停留在“>”处，我们可以继续输入网站名称进行下一次查询，这就是交互模式的特点。

如何进入非交互模式

非交互模式，其实并不需要进入，使用“进入”这个词，或许会有些误导之嫌。

如果你直接在 nslookup 命令后加上所要查询的域名，那么这就是非交互模式了。

比如，我们希望在非交互模式下查询 www.baidu.com 域名对应的 IP 地址：

```
#查询www.baidu.com域名对应的IP地址
[roc@roclinux ~]$ nslookup www.baidu.com
Server:        223.5.5.5
Address:       223.5.5.5#53

Non-authoritative answer:
www.baidu.com   canonical name = www.a.shifen.com.
Name:   www.a.shifen.com
Address: 61.135.169.125
Name:   www.a.shifen.com
Address: 61.135.169.121

[roc@roclinux ~]$
```

可以看到，在非交互模式下，我们进行完一次查询后，提示符会回到 Shell 提示符状态，下一次再想查询 DNS 信息的话还需要执行 nslookup 命令，这就是非交互模式的特点。

好了，本文为大家介绍了 nslookup 的两种工作模式，并且演示了域名查询的方法，以及指定域名服务器的方法。要想深入掌握 nslookup，那就一定要对 DNS 协议有比较深入的理解，所以，在接下来的一篇文章中，我们来学习 DNS 协议，对于大部分同学来说，可能算温故而知新啦。

3 DNS 探秘之二——DNS 知识温故知新

在介绍完 nslookup 的模式之后，我们就要深入到其命令输出部分了。这些内容是和 DNS 协议的知识紧密相关的，所以在讲解之前，我们需要把 DNS 的各类知识提前讲解温习一下，确保大家可以更好更顺畅地理解 nslookup 输出内容。

说到 DNS 协议，相信大家都知道它是一个应用层的协议，大部分场景下是用来管理域名和 IP 地址的映射关系的。

DNS 这套系统，采用了一种多级、分层次的组织方案，来实现对域名信息的管理。它的工作过程，可以简述如下：

1. 为了解析一个域名（如 www.baidu.com），应用程序会调用域名解析库函数，并将域名作为参数传入其中。
2. 然后，此解析库函数会提取本机所设置的上连 DNS 服务器地址，并向此地址发送一个域名解析请求。
3. 上连 DNS 服务器接到域名解析请求后，在本机查找此域名，找到后将其对应的 IP 地址返回给解析库函数。
4. 解析库函数接收到上连 DNS 服务器返回的信息后，将此信息返回给应用程序。

通过上面的 4 个步骤，应用程序就可以拿到一个域名对应的 IP 地址信息了。

而理想很丰满，现实很骨感，域名的解析过程并非总是那么顺利，有时候，我们会遇到“上连 DNS 服务器”在它的本机并没有搜索到你要查询的域名信息的情况，这又如何是好呢？

DNS 的递归查询是怎样的

为了更清楚地为大家介绍 DNS 查询的过程，也为了解答上一小节的那个遗留问题，我们来做一个情景假设。

北邮（北京邮电大学）计算机学院的官方网站是 scs.bupt.edu.cn，其中：

- cn：是国家域名。
- edu：是教育行业专属域名。
- bupt：是北京邮电大学的英文缩写，即 Beijing University of Post and Telecommunication。
- scs：是计算机学院的缩写，即 School of Computer Science。

假如小吴在家中希望访问北邮计算机学院的官方网站 scs.bupt.edu.cn，于是浏览器会首先发起对这个域名的 DNS 解析，我们来一起看看域名解析的过程，如图 1 所示。图中连线处的数字，与下面的步骤编号对应。

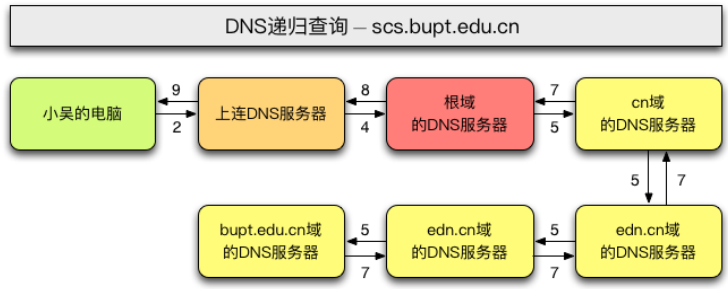


图 1　递归查询过程

第 1 步：为了解析这个域名（scs.bupt.edu.cn），浏览器会调用域名解析库函数，并将域名作为参数传入其中。

第 2 步：然后，此解析库函数会提取本机所设置的上连 DNS 服务器地址，并向此地址发送一域名解析请求。

第 3 步：上连 DNS 服务器接到域名解析请求后，在本机查找此域名，但是，遗憾的是，它并没有找到对应的信息。

第 4 步：于是上连 DNS 服务器就会求助于根 DNS 服务器（root-servers.net）。

第 5 步：此后，请求会逐级转发，顺序是 cn 的 DNS 服务器→edu.cn 的 DNS 服务器→bupt.edu.cn 的 DNS 服务器。

第 6 步：由于 bupt.edu.cn 一定是管理 scs.bupt.edu.cn 的资源记录的 DNS 服务器，于是，bupt.edu.cn 的 DNS 服务器会将所查询域名对应的 IP 地址返回给上一级。

第 7 步：这个 IP 地址会按照逆序依次回传，顺序是 edu.cn 的 DNS 服务器→cn 的 DNS 服务器→根 DNS 服务器。

第 8 步：最后根 DNS 服务器将这个信息返回给上连 DNS 服务器。

第 9 步：上连 DNS 服务器收到结果后，首先会将这个结果缓存到本机，同时，将其回传给小吴家中的浏览器。

这是一个典型的 DNS 递归查询（Recursive Query）过程，也确实比上一段落中描述的顺利过程复杂了不少。

这里要注意一个细节，那就是上连 DNS 服务器在收到结果后，不仅会把结果返回给调用者，还会自己缓存起来哦。

至于缓存起来之后的用处和可能带来的问题，我们先卖个关子，在深入讲解 nslookup 命令时会揭晓答案。

除了递归还有什么

聪明的小伙伴们，或许已经隐隐感觉到了，既然我们明确指出了 DNS 递归查询过程，那就一定还存在另一种查询过程。是的，你没猜错，DNS 还为我们提供另一种可选方案，那就是迭代查询。迭代查询的过程是这样的，我们再次回到小吴的浏览器面前，如图 2 所示。

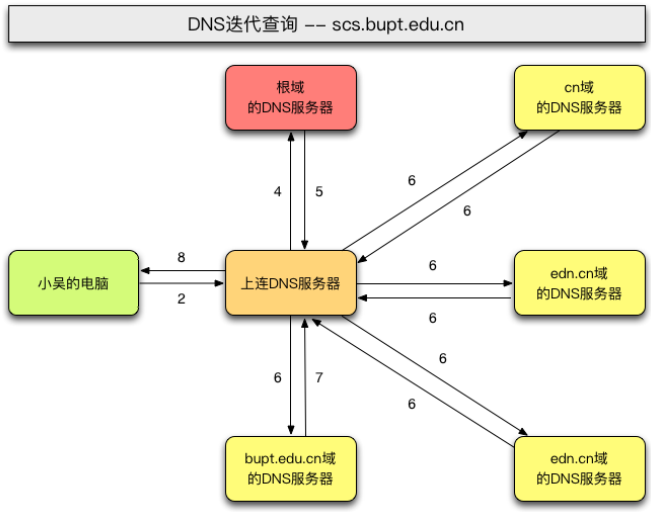


图 2　迭代查询过程

第 1 步：为了解析这个域名（scs.bupt.edu.cn），浏览器仍然会调用域名解析库函数，并将域名作为参数传入其中。

第 2 步：然后，此解析库函数会提取本机所设置的上连 DNS 服务器地址，并向此地址发送一个域名解析请求。

第 3 步：上连 DNS 服务器接到域名解析请求后，在本机查找此域名，但是，遗憾的是，它并没有找到对应的信息。

第 4 步：于是上连 DNS 服务器就会求助于根 DNS 服务器，注意迭代查询要开始啦。

第 5 步：根 DNS 服务器会将 cn 的 DNS 服务器的地址返回给上连 DNS 服务器，潜台词就是“虽然我是根 DNS 服务器，但我也不是万能的，我其实也没法直接告诉你答案，但是呢，我可以给你引荐一位高人，它一定可以告诉你进一步的线索，它就是 cn 的 DNS 服务器，你去问问它吧，回见！”

第 6 步：就这样，上连 DNS 服务器又依次去拜访了 cn 的 DNS 服务器→edu.cn 的 DNS 服务器→bupt.edu.cn 的 DNS 服务器。

第 7 步：由于 bupt.edu.cn 一定是知道 scs.bupt.edu.cn 的资源记录的，所以 bupt.edu.cn 的 DNS 服务器终于把最终答案告诉了上连 DNS 服务器。

第 8 步：上连 DNS 服务器收到结果后，首先会将这个结果缓存到本机，同时，将其回传给小吴家中的浏览器。

这就是一个完整的 DNS 迭代查询过程，它和递归查询的最大区别在于：

- 递归查询：好事办到底，每个环节都承担起了寻找答案的责任，都会负责去询问它的下一级 DNS 服务器。
- 迭代查询：给你指路，每个环节都是告诉你下一步怎么走，路还是要自己去走。

除了递归和迭代还有什么

那么，问题来了，除了递归查询、迭代查询，DNS 查询还有什么新花样呢？真的还有，那就是“递归与迭代相结合”的查询方法。

这种查询方法产生的背景是，根 DNS 服务器在全球仅有 13 台，每天都承载着全

球数以亿计的查询请求。为了尽量降低它们的工作负载，它们在收到查询请求后，会采用迭代查询的方法，将下一级 DNS 服务器地址返回给请求方，而请求方再次去请求下一级 DNS 服务器时，则开始采用递归查询的方法。这样，可以有效地降低根 DNS 服务器的负荷，提高全球的 DNS 查询效率，如图 3 所示。相信大家可以自己看懂下图。

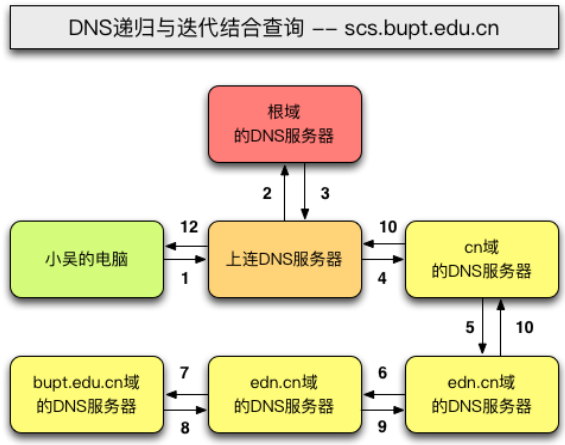


图 3　递归与迭代结合的查询过程

域名解析的缓存

还记得刚才我们在讲解递归查询时卖了一个关子么，有关“缓存”的关子，现在我们就来揭晓答案咯。

如果你仔细看了 nslookup 的输出内容的话，就会发现里面有“Non-authoritative answer”的字样，就像这样：

```
[roc@roclinux ~]$ nslookup www.baidu.com
Server:         223.5.5.5
Address:        223.5.5.5#53

Non-authoritative answer:
www.baidu.com   canonical name = www.a.shifen.com.
Name:   www.a.shifen.com
Address: 61.135.169.121
Name:   www.a.shifen.com
Address: 61.135.169.125
```

看到了吧，这个“Non-authoritative answer”就是和缓存知识密切相关的，大家只有了解了缓存的原理和知识后，才能更好地理解“Non-authoritative answer”的含义。

现在我们就来说说有关 DNS 服务器的缓存的话题，这里面的学问还真的不少呢。

每一个 DNS 服务器都有一个高速缓存区，这里面存放着这台 DNS 服务器最近经手过的“域名—IP 地址”映射关系，以及获得这些映射关系的查询出处。正如我们上例中所描述的，上连 DNS 服务器在获得了 scs.bupt.edu.cn 的 IP 地址后，就会把这个映射关系缓存在自己机器上，这样做的好处就在于，下次再有人询问 scs.bupt.edu.cn 的 IP 信息的话，这台上连 DNS 服务器就不需要再发起递归查询或迭代查询了，直接就可以从缓存中找到答案，并快速返回给询问者了。

理想很丰满，现实又是很骨感。我又要给大家泼冷水了。你有没有想过，事情其实并没有那么顺利，因为机器经常会发生故障，网站也会偶尔迁移，所以域名对应的 IP 地址是会变化的。所以呢，DNS 服务器不能把这个信息缓存一辈子，需要给它设定一个期限，当然不可能是一万年，一般会是小时级别的期限。一旦超过这个期限，DNS 服务器就会不再信任自己缓存中的信息，而是重新发起迭代查询或递归查询，去寻求最新的答案。

有了这样的机制，是不是我们的 DNS 查询就完美了呢，答案是 NO！虽然我们将期限设置为了小时级，但是仍然会存在不走运的“时间差”。假如一台 DNS 服务器在 14:00 刚刚缓存了 scs.bupt.edu.cn 的解析数据，很不幸在 14:05 分，scs.bupt.edu.cn 因为网站迁移修改了其对应的 IP 地址，假如正好有一个用户通过这台 DNS 服务器来查询 scs.bupt.edu.cn 时，就会拿到过时的 IP 地址了，是不是很坑人呢？

正是由于存在这种隐患和时间差，所以，如果 DNS 服务器是从自己缓存中提取解析数据返回给用户的，那么就会很谨慎的在返回内容中给出“Non-authoritative answer”的字样，潜台词就是提示用户：“朋友，我把答案告诉你了，但是，我可不保证它是最新的哦，因为这也是我前一阵听说的，不是我现去查的。”

终于，我们揭开了“Non-authoritative answer”的神秘面纱。

不过，又会有好奇的同学提问了：“Non-authoritative 在英文中是非权威的意思，这个词用得怪怪的，有什么讲究么？”

授权和非授权

还真的是有讲究的。还记得刚才的迭代查询的例子么，例子中的 DNS 服务器会为你指一条明路，即“虽然我不知道 XXX 的信息，但是，我可以给你引荐一位高人，

他应该可以给你进一步的线索。”是的！这位高人，就是我们下面要讲的“被授权的人”。

在 DNS 的世界里，没有人知道全部答案，但人们总能告诉你离正确答案更近的下一位高人。

我们来举个例子：

虽然 edu.cn 的 DNS 服务器（我们称为 A 机）自己并不知道 scs.bupt.edu.cn 的 IP 地址，但是，edu.cn 的 DNS 服务器知道负责 bupt.edu.cn 域名解析的 DNS 服务器（我们称为 B 机）的地址，B 机就是那个离正确答案更近的高人。

所以，edu.cn 的 DNS 服务器（A 机）就将 bupt.edu.cn 的域名解析授权给了这台 DNS 服务器（B 机）。只有这台 DNS 服务器（B）所返回的有关 scs.bupt.edu.cn 的域名解析，才是授权的；否则，就称为非授权的。

一般情况下，一台 DNS 服务器从自身缓存中提取结果并返回给询问者，这个结果就是非权威（Non-authoritative）的。

好了，通过这样一篇文章我们把 DNS 的几个重要概念都复习了一遍，这为我们阅读后面的文章奠定了良好的基础，相信大家阅读接下来的文章时会更加顺畅。接下来，我们就开始为大家解读 nslookap 的输出内容。

4 DNS 探秘之三——nslookup 输出解析

我们仍然以 www.baidu.com 的域名解析来作为样例，为大家详细解释 nslookup 的输出中都包含了哪些重要内容：

```
[roc@roclinux ~]$ nslookup www.baidu.com
Server:        223.5.5.5
Address:       223.5.5.5#53

Non-authoritative answer:
www.baidu.com   canonical name = www.a.shifen.com.
Name:   www.a.shifen.com
Address: 61.135.169.121
Name:   www.a.shifen.com
Address: 61.135.169.125
```

可以看出，nslookup 的输出包含了上下两部分。

- 上半部分：DNS 服务器信息。
- 下半部分：域名解析信息。

DNS 服务器信息中包含了两行内容，就是下面两行：

```
Server:        223.5.5.5
Address:       223.5.5.5#53
```

第一行的 Server，表示我们本次 DNS 解析所使用的 DNS 服务器名称，默认会采用系统里/etc/resolv.conf 文件中所配置的第一个 DNS 服务器。

第二行的 Address，表示我们连接到的 DNS 服务器的具体 IP 地址和端口。这里#53 就表示我们访问的是 DNS 服务器的 53 号端口。53 号端口也是 DNS Server 的默认服务端口。在亚马逊云 AWS 中就有一个独立的产品叫作 Route53，它是一个强调高可用和灵活伸缩的域名系统（DNS）服务，名字中就直接使用了 53 这个数字。

介绍完了 nslookup 输出中的上半部分，我们再来看看重头戏，nslookup 输出中的下半部分，也就是“域名解析信息”：

```
Non-authoritative answer:
www.baidu.com   canonical name = www.a.shifen.com.
```

```
Name:   www.a.shifen.com
Address: 61.135.169.121
Name:   www.a.shifen.com
Address: 61.135.169.125
```

首先，映入眼帘的便是“Non-authoritative answer”，通过前面的学习我们知道，这表示返回的结果是非权威的，可能会因为时间差的原因而不是最新的。

而下一行的信息，是我们之前没见过的，即：

```
www.baidu.com   canonical name = www.a.shifen.com.
```

canonical name，就是我们常说的 CNAME，也就是别名。所以我们可以看到，www.baidu.com 背后的别名是 www.a.shifen.com。

再下面的内容，则表示真正的答案，也就是 www.baidu.com 所对应的两个 IP 地址：

```
Name:   www.a.shifen.com
Address: 61.135.169.121
Name:   www.a.shifen.com
Address: 61.135.169.125
```

本文内容，相对来说比较紧凑精炼，和大家一起解读了 nslookup 的输出内容，将来你也可以为别人讲解这些知识，享受别人投来的羡慕目光。是不是想想就觉得很美好，哈哈！

5 DNS 探秘之四——DNS 协议中的五元组

传说中的五元组

一提到 DNS，相信大家都会直观地认为 DNS 就是用来做域名和 IP 地址转换的，但其实并非如此，准确地说，DNS 是用来做域名和资源转换的，而 IP 地址只是资源中的一种而已。

当我们把一个域名查询的请求传递给 DNS 服务器后，DNS 服务器所返回的是与该域名相关的资源记录。那么资源是个什么东西呢？

在 DNS 的世界里，资源是一个五元组，即：

```
{ DomainName、TimeToLive、Class、Type、Value }
```

翻译成中文便是：

```
{ 域名、生存期限、类别、类型、值 }
```

它们的含义如下。

- DomainName（域名）：指我们要查询的那个域名。
- TimeToLive（生存期限）：表示此域名在各 DNS 服务器缓存中应保存的时长。
- Class（类别）：通常为 IN，即 Internet。另外还有 CH（Chaos）和 HS（Hesiod）两类，但目前几乎已经被淘汰了。
- Type（类型）：指出这条记录的类型，包括 8 种，即 SOA、A、MX、NS、CNAME、PTR、HINFO 和 TXT。
- Value（值）：针对不同类型，会有不同的值。

为了让大家有一个直观的认识，我们截取了 dig 命令（后面的文章会讲到）的一小段输出作为例子：

```
scs.bupt.edu.cn.        7200    IN      CNAME   revp.bupt.edu.cn.
revp.bupt.edu.cn.       7200    IN      A       124.127.207.100
```

因为我们已经介绍了五元组，所以这两行内容，相信大家已经大概看懂了：

- 第 1 行：表示 scs.bupt.edu.cn 是 revp.bupt.edu.cn 的别名，其超时时限是 7200 秒。
- 第 2 行：表示 revp.bupt.edu.cn 对应的 IP 地址是 124.127.207.100，其超时时限同样为 7200 秒。

DNS 的八种类型

学习 DNS 五元组中的 type 知识，对于理解和使用 nslookup 命令是很重要的，在上一段落中，我们知道 type 包含了 8 个可选类型，即：

- SOA：Start of Authority，授权起始。
- A：IP 地址。
- MX：邮件交换。
- NS：域名服务器。
- CNAME：别名，也叫规范名。
- PTR：指针，用于反向解析。
- HINFO：主机描述信息，包括 CPU 和 OS 等信息。
- TXT：其他一些文本信息。

为了让后面的内容学习起来更加顺畅，有必要为大家提前介绍一下这 8 种类型。

类型 SOA

SOA，即 Start Of Authority，表示授权起始。在这里，我们可以获得针对一个域名的最基本的设置信息，包括但不限于：

- Mail：管理员邮箱地址。
- Serial：版本序号，格式一般为年月日次，如 2016031903 则表示 2016 年 03 月 19 日的第 03 个版本。
- Refresh Slave：表示 Slave 的 DNS 服务器多久向 Master 的 DNS 服务器要一次更新数据。
- Retry：在发起 Refresh 时，如果 Slave 连接不到 Master，那么间隔多久进行一次连接尝试。
- Expire：在发起 Refresh 时，如果 Slave 始终无法连接到 Master，那么多久后放弃尝试。

- Minimum：即 TTL，表示外部 DNS 服务器如果要缓存本 DNS 服务器的授权数据，那么保存时限是多久。

```
[roc@roclinux ~]$ nslookup -type=soa bupt.edu.cn
Server:        223.5.5.5
Address:       223.5.5.5#53

Non-authoritative answer:
bupt.edu.cn
       origin = ns.buptnet.edu.cn
       mail addr = admin.bupt.edu.cn
       serial = 2016031675
       refresh = 10800
       retry = 900
       expire = 604800
       minimum = 86400

Authoritative answers can be found from:

[roc@roclinux ~]$
```

类型 A

A，表示从域名解析到 IP 地址。此处用来展示其对应的 IP 地址信息，俗称“A 记录”。

```
[roc@roclinux ~]$ nslookup -type=a www.bupt.edu.cn
Server:        223.5.5.5
Address:       223.5.5.5#53

Non-authoritative answer:
Name:   www.bupt.edu.cn
Address: 114.255.40.66
```

类型 MX

MX，是 Mail eXchanger 的缩写，即邮件交换。此类型和邮箱服务器设置有关，用来表示当前域名对应的邮箱服务器。

```
 [roc@roclinux ~]$ nslookup -type=mx bupt.edu.cn
Server:        223.5.5.5
Address:       223.5.5.5#53

Non-authoritative answer:
bupt.edu.cn    mail exchanger = 5 mx3.bupt.edu.cn.
bupt.edu.cn    mail exchanger = 5 mx1.bupt.edu.cn.
bupt.edu.cn    mail exchanger = 5 mx2.bupt.edu.cn.

Authoritative answers can be found from:

[roc@roclinux ~]$
```

当然，如果你的域名没有配置对应的邮箱服务器，则 MX 为空。

类型 NS

NS，即 Name Server，表示给定域名下所包含的 DNS 服务器信息。

```
[roc@roclinux ~]$ nslookup -type=ns bupt.edu.cn
Server:        223.5.5.5
Address:       223.5.5.5#53

Non-authoritative answer:
bupt.edu.cn    nameserver = ns.buptnet.edu.cn.
bupt.edu.cn    nameserver = gus.buptnet.edu.cn.

Authoritative answers can be found from:

[roc@roclinux ~]$
```

类型 CNAME

CNAME，即 Canonical Name，也叫别名。比如 scs.bupt.edu.cn 就是 revp.bupt.edu.cn 的一个别名。

```
[roc@roclinux ~]$ nslookup -type=cname scs.bupt.edu.cn
Server:        223.5.5.5
Address:       223.5.5.5#53

Non-authoritative answer:
scs.bupt.edu.cn canonical name = revp.bupt.edu.cn.

Authoritative answers can be found from:

[roc@roclinux ~]$
```

类型 PTR

PTR，即指针，用来表示反解信息，即从 IP 地址查询其对应域名的映射关系。

类型 HINFO

HINFO，会包含 CPU 和 OS 等信息。

类型 TXT

TXT，即文本信息，表示有关此域名的一些信息。

好了，DNS 五元组的知识就讲到这里了，虽然有些枯燥，但是对于理解 nslookup 命令和接下来的 dig 命令都是非常有帮助的。

6 DNS 探秘之五——nslookup 交互模式

直接查询域名 A 记录

在掌握了那么多的 DNS 知识之后，我们重新进入到 nslookup 的交互模式中，一起再温习一遍各种 DNS 查询的用法。

如果你直接输入域名（以 www.baidu.com 举例），会有类似如下的输出：

```
$ nslookup
> www.baidu.com
Server:         61.139.2.69 //上连的DNS服务器
Address:        61.139.2.69#53 //上连的DNS服务器的IP地址与端口号

Non-authoritative answer: //非权威答案，即从上连DNS服务器的本地缓存中读取的值
www.baidu.com   canonical name = www.a.shifen.com. //说明www.baidu.com有别名叫www.a.shifen.com
Name:   www.a.shifen.com //域名www.a.shifen.com
Address: 119.75.217.56 //对应的IP地址之一
Name:   www.a.shifen.com
Address: 119.75.218.77//对应的IP地址之二
```

更改上连 DNS 地址

默认情况下，nslookup 会连接到本机的默认上连 DNS 服务器去查询域名的 IP 地址。

其实，我们可以使用 server 命令来指定上连 DNS 服务器的地址，来看下面的例子：

```
$ nslookup
> www.baidu.com //以默认的上连DNS服务器来查询
Server:         61.139.2.69
Address:        61.139.2.69#53

Non-authoritative answer:
www.baidu.com   canonical name = www.a.shifen.com.
Name:   www.a.shifen.com
Address: 119.75.218.77
Name:   www.a.shifen.com
Address: 119.75.217.56
```

```
> server 8.8.8.8 //更改了上连的DNS服务器地址为谷歌开放DNS
Default server: 8.8.8.8
Address: 8.8.8.8#53
> www.baidu.com //以更改后的上连DNS服务器来查询
Server:        8.8.8.8
Address:       8.8.8.8#53

Non-authoritative answer:
www.baidu.com  canonical name = www.a.shifen.com.
Name:  www.a.shifen.com
Address: 220.181.111.147 //看！IP地址变化了，说明百度采用了智能DNS解析策略
```

从这个例子可以得到一个知识，那就是，针对同一个域名，两个不同的上连 DNS 服务器返回的 IP 地址有可能是不同的。这可能是由于域名本身采用了智能 DNS 解析策略所造成的。

查看当前 DNS 的配置

列出 nslookup 工具的常用选项的当前设置值。

```
> set all
Default server: 8.8.8.8 //当前的上连DNS服务器
Address: 8.8.8.8#53 //当前的上连DNS服务器的IP地址和端口

Set options:
  novc               nodebug        nod2
  search             recurse
  timeout = 0         retry = 3      port = 53
  querytype = A       class = IN
  srchlist =
```

在上面所列出的配置项中，大家可以读懂大部分内容，包括 timeout、querytype、class、retry、port 等，但有些配置项，比较高级，而在本系列文章中，不会继续介绍这些高级配置项，有兴趣的读者可以通过计算机网络的专业书籍来学习和研究。

进入调试模式，查看更多交互信息

通过开启 debug 开关，可以让我们进入调试模式。在调试模式下，域名查询过程中完整的响应包及其中的交互包，都会显示出来，方便大家追查问题和调试功能。

```
$ nslookup
> set debug //进入调试模式
> www.baidu.com
Server:        61.139.2.69
Address:       61.139.2.69#53
```

```
------------
    QUESTIONS: //发出的查询请求
       www.baidu.com, type = A, class = IN
    ANSWERS: //返回的信息
    ->  www.baidu.com
       canonical name = www.a.shifen.com.
       ttl = 1192
    ->  www.a.shifen.com
       internet address = 119.75.217.56
       ttl = 262
    ->  www.a.shifen.com
       internet address = 119.75.218.77
       ttl = 262
    AUTHORITY RECORDS:
    ADDITIONAL RECORDS:
------------
Non-authoritative answer:
www.baidu.com   canonical name = www.a.shifen.com.
Name:   www.a.shifen.com
Address: 119.75.217.56
Name:   www.a.shifen.com
Address: 119.75.218.77
```

注意：如果你使用 set d2 命令，则会开启高级调试模式，会输出更多 nslookup 内部工作的信息，甚至包括许多函数调用信息。

指定查询中默认的域后缀

设置默认的域后缀，可以为我们的域名查询带来便捷。因为，对于所有尾部不包含“.”的查询请求，都会自动在尾部追加上我们所设置的域后缀。

```
$ nslookup
> set all //首先显示上连DNS服务器信息以及所有的当前选项信息
Default server: 61.139.2.69
Address: 61.139.2.69#53
Default server: 8.8.8.8
Address: 8.8.8.8#53
Default server: 202.102.224.68
Address: 202.102.224.68#53

Set options:
  novc              nodebug        nod2
  search            recurse
  timeout = 0        retry = 3      port = 53
  querytype = A       class = IN
  srchlist =  //可以看到srchlist为空
> set domain=baidu.com //设置默认域为baidu.com
> set all //再次查看选项信息
Default server: 61.139.2.69
```

```
Address: 61.139.2.69#53
Default server: 8.8.8.8
Address: 8.8.8.8#53
Default server: 202.102.224.68
Address: 202.102.224.68#53

Set options:
  novc               nodebug        nod2
  search             recurse
  timeout = 0         retry = 3      port = 53
  querytype = A       class = IN
  srchlist = baidu.com //可以看到srchlist已经被设置了baidu.com后缀
> image //直接查询image
Server:        61.139.2.69
Address:       61.139.2.69#53

Non-authoritative answer:
image.baidu.com canonical name = image.n.shifen.com. //可以看到已默认追加了.baidu.com域后缀，变成了image.baidu.com
Name:   image.n.shifen.com
Address: 220.181.111.131
```

在交互模式下设置 type

通过 set querytype=[value]，我们可以更改信息查询的类型。

默认情况下，nslookup 是查询域名所对应的 A 记录，而当你想查询其对应的 MX 记录等信息时，就需要更改查询的类型了。

目前常用的 type 值如下：

```
A：查看主机的IPv4地址
AAAA：查看主机的IPv6地址
ANY：查看关于主机域的所有信息
CNAME：查找与别名对应的正式名字
HINFO：查找主机的CPU与操作系统类型
MINFO：查找邮箱信息
MX：查找邮件交换信息
NS：查找主机域的域名服务器
PTR：查找与给定IP地址匹配的主机名
RP：查找域负责人记录
SOA：查找域内的SOA地址
UINFO：查找用户信息
```

例如，针对 MX 类型的查询结果如下：

```
> set type=MX
> baidu.com //查询MX信息
```

```
Server:        61.139.2.69
Address:       61.139.2.69#53

Non-authoritative answer:
baidu.com      mail exchanger = 20 jpmx.baidu.com.
baidu.com      mail exchanger = 20 mx50.baidu.com.
baidu.com      mail exchanger = 10 mx.mailcdn.baidu.com.
baidu.com      mail exchanger = 20 mx1.baidu.com.

Authoritative answers can be found from:
> set type=A
> baidu.com //查询A记录
Server:        61.139.2.69
Address:       61.139.2.69#53

Non-authoritative answer:
Name:   baidu.com
Address: 123.125.114.144
Name:   baidu.com
Address: 220.181.111.85
Name:   baidu.com
Address: 220.181.111.86
```

好了，关于 nslookup 命令的相关知识我们就介绍到这里了。接下来的两篇文章，我们会为大家介绍另一个好用的 DNS 查询工具 dig。

7 DNS 探秘之六——dig 初体验

dig 为自己代言

在平时查询 DNS 信息时，相信大家使用 nslookup 会更多一些，对 dig 深入了解的同学相对较少，所以借这个机会，我们有必要花些时间给大家介绍一下 dig 命令的功能和用法。很多人不知道，其实 dig 要比 nslookup 更加强大，dig 是个有颜值的实力派。

对了，如果你还没有阅读过本系列的前几篇文章，那么我建议你还是先去读一下，那里面介绍了不少有关 DNS 的基础知识，对于大家理解 dig 有很大帮助。

dig，其实是一个缩写，即 Domain Information Groper，它是一个 DNS 查询工具。

一些专业的 DNS 管理员在追查 DNS 问题时，都乐于使用 dig 命令，看中的便是 dig 设置灵活、输出清晰、功能强大的特点。

最简单的 dig 用法

最简单的 dig 用法，当然就是直接执行 dig 命令啦：

```
[roc@roclinux ~]$ dig

; <<>> DiG 9.8.2rc1-RedHat-9.8.2-0.37.rc1.el6_7.4 <<>>
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 33541
;; flags: qr rd ra; QUERY: 1, ANSWER: 13, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;.                          IN     NS

;; ANSWER SECTION:
.                   197458  IN     NS     j.root-servers.net.
.                   197458  IN     NS     b.root-servers.net.
.                   197458  IN     NS     f.root-servers.net.
.                   197458  IN     NS     l.root-servers.net.
.                   197458  IN     NS     h.root-servers.net.
```

```
.                       197458  IN      NS      c.root-servers.net.
.                       197458  IN      NS      k.root-servers.net.
.                       197458  IN      NS      i.root-servers.net.
.                       197458  IN      NS      e.root-servers.net.
.                       197458  IN      NS      m.root-servers.net.
.                       197458  IN      NS      g.root-servers.net.
.                       197458  IN      NS      a.root-servers.net.
.                       197458  IN      NS      d.root-servers.net.

;; Query time: 43 msec
;; SERVER: 223.5.5.5#53(223.5.5.5)
;; WHEN: Tue Mar  1 11:20:31 2016
;; MSG SIZE  rcvd: 228
```

从上面的输出可以发现，当直接使用 dig 命令，不加任何参数和选项时，dig 会向上连 DNS 服务器查询“.”（根域）的 NS 记录。

dig 后加个点

刚才是直接输入 dig，这次我们在后面加上一个点号“.”，表示查询根域，看看结果和刚才相比有什么区别：

```
[roc@roclinux ~]$ dig .

; <<>> DiG 9.8.2rc1-RedHat-9.8.2-0.37.rc1.el6_7.4 <<>> .
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 20722
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 0

;; QUESTION SECTION:
;.                          IN      A

;; AUTHORITY SECTION:
.                       9820    IN      SOA     a.root-servers.net.
nstld.verisign-grs.com. 2016022901 1800 900 604800 86400

;; Query time: 56 msec
;; SERVER: 223.5.5.5#53(223.5.5.5)
;; WHEN: Tue Mar  1 11:21:12 2016
;; MSG SIZE  rcvd: 92
```

可以看出，这次输出的是根域的 A 记录，这个例子中展示了 13 个根域之一的 a.root-servers.net.的五元组信息。

我们在前面的文章中介绍过“DNS 的五元组知识”，大家可以回去温故知新一下。

我想用谷歌开放 DNS 来查询

在 2009 年，Google 对外发布了一项全新的服务，即谷歌公共域名解析服务，英文称之为 Google Public DNS，旨在为全球用户提供安全可靠的 DNS 解析服务。它的首选 DNS 服务器是 8.8.8.8，而备选 DNS 服务器则为 8.8.4.4，都是非常好记的数字串。

下面这个例子，就是使用谷歌开放 DNS 服务器来查询百度域名的 A 记录：

```
$ dig @8.8.8.8 www.baidu.com A //命令格式为dig @dnsserver name querytype

;  <<>>  DiG  9.8.2rc1-RedHat-9.8.2-0.37.rc1.el6_7.4  <<>>  @8.8.8.8
www.baidu.com A
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 48548
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.baidu.com.               IN     A

;; ANSWER SECTION:
www.baidu.com.         1185    IN     CNAME   www.a.shifen.com.
www.a.shifen.com.       192     IN     A       103.235.46.39

;; Query time: 87 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Tue Mar  1 11:26:03 2016
;; MSG SIZE  rcvd: 74
```

从上面这个例子，大家学习到了 dig 的基本的命令格式是：

```
$ dig @dnsserver name querytype
```

如果你没有设置@dnsserver，那么 dig 会依次使用/etc/resolv.conf 里的地址作为上连 DNS 服务器。

而对于 querytype，如果你阅读了《DNS 探秘之四》和《DNS 探秘之五》两篇文章，那么你应该对 querytype 有所了解。如果忘记了，就赶快回去复习一下吧。

8 DNS 探秘之七——dig 选项走马观花

用-f 选项实现批量查询

dig 命令支持从一个文件里读取内容进行批量查询，这个功能非常体贴。我们来看一个实际的例子：

```
#文件内容，共有两个域名需要查询
[roc@roclinux ~]$ cat querylist
www.baidu.com
www.sohu.com

#设置-f参数开始批量查询，同时通过-t来指定查询类型
[roc@roclinux ~]$ dig -f querylist -t A

; <<>> DiG 9.8.2rc1-RedHat-9.8.2-0.37.rc1.el6_7.4 <<>> www.baidu.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 4725
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.baidu.com.                 IN      A

;; ANSWER SECTION:
www.baidu.com.          234     IN      CNAME   www.a.shifen.com.
www.a.shifen.com.       234     IN      A       112.80.248.74
www.a.shifen.com.       234     IN      A       112.80.248.73

;; Query time: 37 msec
;; SERVER: 223.5.5.5#53(223.5.5.5)
;; WHEN: Tue Mar  1 15:24:38 2016
;; MSG SIZE  rcvd: 90

; <<>> DiG 9.8.2rc1-RedHat-9.8.2-0.37.rc1.el6_7.4 <<>> www.sohu.com
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 54774
;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.sohu.com.                  IN      A

;; ANSWER SECTION:
```

```
www.sohu.com.          28      IN      CNAME   gs.a.sohu.com.
gs.a.sohu.com.         28      IN      CNAME   fbjuni.a.sohu.com.
fbjuni.a.sohu.com.      28      IN      A       61.135.132.52
fbjuni.a.sohu.com.      28      IN      A       123.125.116.12

;; Query time: 32 msec
;; SERVER: 223.5.5.5#53(223.5.5.5)
;; WHEN: Tue Mar  1 15:24:38 2016
;; MSG SIZE  rcvd: 102
```

用-t 选项设置查询类型

我们可以使用-t 选项设置要查询的资源类型，默认情况下是 A。如果我们想查看域名的 MX 信息，则方法如下：

```
#我们设置查询类型为MX，即邮箱服务器
[roc@roclinux ~]$ dig roclinux.cn -t MX

; <<>> DiG 9.8.2rc1-RedHat-9.8.2-0.37.rc1.el6_7.4 <<>> roclinux.cn -t MX
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 64828
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;roclinux.cn.                   IN      MX

;; ANSWER SECTION:
roclinux.cn.            3600    IN      MX      0 mail.roclinux.cn.

;; Query time: 522 msec
;; SERVER: 223.5.5.5#53(223.5.5.5)
;; WHEN: Tue Mar  1 15:25:47 2016
;; MSG SIZE  rcvd: 50
```

从查询结果中可以看到，Linux 大棚博客的邮箱服务器地址是 mail.roclinux.cn。

看似没用却有用的-q 选项

说到-q 选项，其实它本身是一个多余的选项，但是它在复杂的 dig 命令中又特别有用。好像前言后语有些自相矛盾，哈哈，请大家继续往后看。

-q 选项可以显式地设置你要查询的域名，这样可以避免和其他众多的参数、选项相混淆，提高了命令的可读性，我们来看一个例子：

```
[roc@roclinux ~]$ dig -q www.roclinux.cn -t A

;    <<>>    DiG    9.8.2rc1-RedHat-9.8.2-0.37.rc1.el6_7.4    <<>>    -q
```

```
www.roclinux.cn
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 48616
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.roclinux.cn.             IN     A

;; ANSWER SECTION:
www.roclinux.cn.       3600    IN     A      116.255.245.206

;; Query time: 512 msec
;; SERVER: 223.5.5.5#53(223.5.5.5)
;; WHEN: Tue Mar  1 15:26:16 2016
;; MSG SIZE  rcvd: 49
```

虽然输出内容并没有什么变化，但是这样的写法可以让人很清晰地看到我们要查询的是 www.roclinux.cn 域名。有些同学可能还是会云里雾里，没关系，因为你还没有碰到那些极短且极易与参数混淆的域名。

所以，为了提高命令可读性，建议大家养成使用-q 选项的好习惯。

用-x 选项实现反解

-x 选项是逆向查询选项，也称为反解选项。反解功能，可以查询 IP 地址到域名的映射关系。举一个例子：

```
[roc@roclinux ~]$ dig -x 193.0.14.129

; <<>> DiG 9.8.2rc1-RedHat-9.8.2-0.37.rc1.el6_7.4 <<>> -x 193.0.14.129
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 35204
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;129.14.0.193.in-addr.arpa.     IN     PTR

;; ANSWER SECTION:
129.14.0.193.in-addr.arpa. 4493 IN     PTR    k.root-servers.net.

;; Query time: 31 msec
;; SERVER: 223.5.5.5#53(223.5.5.5)
;; WHEN: Tue Mar  1 15:26:47 2016
;; MSG SIZE  rcvd: 75
```

可以看到，这个 IP 地址对应的是一台根域服务器，即 k.root-servers.net，我们成功地实现了 IP 到域名的反解。

dig 特有的查询选项

和刚才的选项不同，dig 还有一批所谓的“查询选项”，这批选项的使用与否，会影响到 dig 的查询方式或输出的结果信息。因此对于这批选项，dig 要求显式地在其前面统一加上一个“+”符号，这样 dig 识别起来会更加方便，同时命令的可读性也会更强。

dig 总共有 42 个查询选项，涉及 DNS 信息的方方面面，如此多的查询选项，本文不会一一赘述，只会挑出最最常用的几个重点讲解。

查询选项——用 TCP 代替 UDP

众所周知，DNS 查询过程中的交互默认是采用 UDP 协议的。如果你希望采用 TCP 协议来进行 DNS 通信，那么就需要使用+tcp 选项了：

```
[roc@roclinux ~]$ dig +tcp @8.8.8.8 www.baidu.com

; <<>> DiG 9.8.2rc1-RedHat-9.8.2-0.37.rc1.el6_7.4 <<>> +tcp @8.8.8.8
roclinux.cn
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2365
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;roclinux.cn.                IN      A

;; ANSWER SECTION:
roclinux.cn.          3281    IN      A       116.255.245.206

;; Query time: 87 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Tue Mar  1 15:28:55 2016
;; MSG SIZE  rcvd: 45
```

查询选项——默认追加域

大家直接看例子，应该就能理解“默认追加域”的概念了，在例子中我们会使用+domain=somedomain 格式来设置默认域：

```
[roc@roclinux ~]$ dig +domain=baidu.com image
```

```
; <<>> DiG 9.8.2rc1-RedHat-9.8.2-0.37.rc1.el6_7.4 <<>> +domain=baidu.com
image
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 47607
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;image.baidu.com.             IN     A

;; ANSWER SECTION:
image.baidu.com.       202     IN      CNAME   image.n.shifen.com.
image.n.shifen.com.     202     IN      A       123.125.115.60

;; Query time: 24 msec
;; SERVER: 223.5.5.5#53(223.5.5.5)
;; WHEN: Tue Mar  1 15:29:44 2016
;; MSG SIZE  rcvd: 78
```

查询选项——跟踪 dig 全过程

dig 非常著名的一个查询选项就是+trace，当使用这个查询选项后，dig 会从根域查询一直跟踪直到查询到最终结果，并将整个过程信息输出出来。

其他的选项可以忘记，但这个选项可要记住，在追查 DNS 解析问题时，高手都会使用这个选项的。

```
[roc@roclinux ~]$ dig +trace roclinux.cn

; <<>> DiG 9.2.4 <<>> +trace roclinux.cn
;; global options:  printcmd
.                       335937  IN      NS      l.root-servers.net.
.                       335937  IN      NS      b.root-servers.net.
.                       335937  IN      NS      d.root-servers.net.
.                       335937  IN      NS      k.root-servers.net.
.                       335937  IN      NS      h.root-servers.net.
.                       335937  IN      NS      j.root-servers.net.
.                       335937  IN      NS      a.root-servers.net.
.                       335937  IN      NS      e.root-servers.net.
.                       335937  IN      NS      c.root-servers.net.
.                       335937  IN      NS      m.root-servers.net.
.                       335937  IN      NS      g.root-servers.net.
.                       335937  IN      NS      i.root-servers.net.
.                       335937  IN      NS      f.root-servers.net.
//从本地DNS查找到根域DNS列表
;; Received 400 bytes from 10.23.0.231#53(10.23.0.231) in 0 ms

cn.                     172800  IN      NS      c.dns.cn.
cn.                     172800  IN      NS      a.dns.cn.
cn.                     172800  IN      NS      b.dns.cn.
cn.                     172800  IN      NS      e.dns.cn.
```

```
cn.                  172800  IN      NS      ns.cernet.net.
cn.                  172800  IN      NS      d.dns.cn.
//选择了b.root-servers.net这台根域DNS来查找cn.域DNS列表
;; Received 292 bytes from 192.228.79.201#53(b.root-servers.net) in 460
ms

roclinux.cn.           21600   IN      NS      ns11.edong.com.
roclinux.cn.           21600   IN      NS      ns12.edong.com.
//选择了c.dns.cn这台cn.域DNS服务器来查找roclinux.cn的DNS列表
;; Received 76 bytes from 203.119.27.1#53(c.dns.cn) in 0 ms

roclinux.cn.           3600    IN      A       116.255.245.206
roclinux.cn.           3600    IN      NS      ns12.edong.com.
roclinux.cn.           3600    IN      NS      ns11.edong.com.
//最终查找到A记录
;; Received 124 bytes from 61.147.124.145#53(ns11.edong.com) in 104 ms
```

查询选项——精简 dig 输出

我们使用+short 选项，可以让 dig 输出最精简的 CNAME 信息和 A 记录。

```
[roc@roclinux ~]$ dig +short www.baidu.com
www.a.shifen.com.
112.80.248.74
112.80.248.73
```

如果我们想在 dig 的标准输出中精简一些冗余内容，那么可以使用+nocmd +nocomment +nostat 组合选项，这样各类冗余信息就都不会输出了，只会输出最核心的内容：

```
[roc@roclinux ~]$ dig +nocmd +nocomment +nostat www.baidu.com
;www.baidu.com.                IN      A
www.baidu.com.         39      IN      CNAME   www.a.shifen.com.
www.a.shifen.com.       39      IN      A       61.135.169.125
www.a.shifen.com.       39      IN      A       61.135.169.121
```

好了，我们连续七篇的 DNS 探秘系列文章到这里就要结束了，这个系列文章包含的知识点比较多，请大家仔细阅读，多多实践哦。

9 iproute2 系列之一——和 netstat 说再见

什么是 netstat

如果你的电脑上有 Linux 系统，那么直接输入 man netstat，就可以看到 netstat 的帮助信息了。man 对于 netstat 的解释非常言简意赅，只有一句简短地描述：

```
“netstat – Print network connections, routing tables, interface statistics, masquerade connections, and multicast memberships”
```

中文意思是：netstat 可以用来显示网络连接、路由表、接口统计、无效连接和组播成员信息。

从这段简短的描述中，我们可以看出，netstat 有五大作用：

- 显示网络连接信息
- 显示路由信息
- 显示接口统计信息
- 显示无效连接信息
- 显示组播成员信息

为什么要和 netstat 说再见

如果你仔细阅读 man netstat 的内容，就会发现这样一句话：

```
"This program is obsolete."
```

原来 netstat 已经是明日黄花了，官方已经不再更新了。它已经被 ss 命令和 ip 命令所取代，或许在不久的将来，在 Linux 发行版中就见不到 netstat 的身影了。所以，如果还有人在用 netstat，建议赶快去学习 ss 和 ip 命令吧。

我们绘制了一张简单的示意图，来展示具体的替代方案，如图 4 所示。

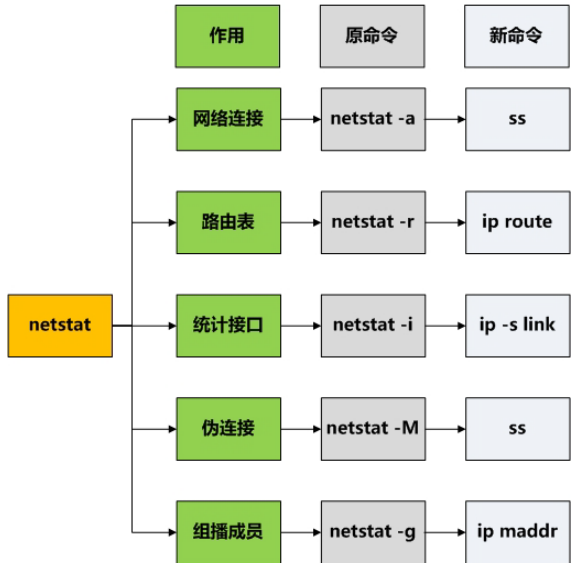


图 4　netstat 替代方案图

接下来我们就为大家介绍 ss 命令和 ip 命令。

10 iproute2 系列之二——篡权的 ss

引言

如果你阅读了前面的“和 netstat 说再见”，那么就应该知道 netstat 已经被无情抛弃的消息了，取而代之的是 ss 命令。有人可能会问“netstat 为什么会被抛弃呢？ss 又是什么命令呢？”

这篇文章，我们就来揭晓答案，为大家介绍这位“篡权的 ss”。

作者粗心大意

ss 命令是一个用来查看 socket 信息的命令，通过 man ss 可以看到，这是由一位俄罗斯人编写的工具，作者的名字显示是 Alexey Kuznetosv，如图 5 所示。

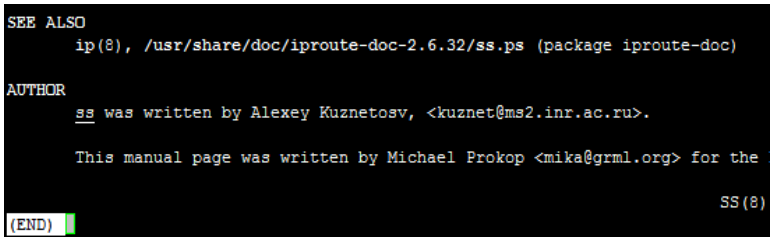


图 5　ss 命令的作者名字

但是，当我们在 Google 上搜索这个名字，以及根据作者提供的邮箱地址所在的网站去查看时，都发现他的名字本应是 Alexey Kuznetsov，区别在于最后的 tsov 或者 tosv。

因为没有学过俄语，所以不知道俄语里是不是本身就允许这样颠倒字母顺序，但是我还是发了封 E-mail 用英语提醒了他一下，看看是不是因为 Alexey 粗心大意，连自己的名字都写错了。

Alexey 目前已经不负责 ss 命令的维护更新了，而是专注在 Linux Kernel QoS 方面

的工作上。

十秒认识 ss

ss 是 Socket Statistics 的缩写。

顾名思义，ss 命令可以用来获取 socket 统计信息，它可以显示和 netstat 类似的内容。但 ss 的优势在于它能够显示更多更详细的有关网络连接状态的信息，而且比 netstat 更快速、更高效。

和 netstat 说再见的原因

当服务器的 socket 连接数量变得非常大时，无论是使用 netstat 命令还是直接读取 /proc/net/tcp 文件，执行速度都会非常慢。当服务器维持的连接达到上万个的时候，使用 netstat 就等于浪费生命，而选用 ss 才是明智的选择。

天下武功唯快不破。ss 快的秘诀在于，它利用到了 TCP 协议栈中的 tcp_diag 模块。tcp_diag 是一个用于分析统计的模块，可以获得 Linux 内核中的第一手信息，这就确保了 ss 的快捷高效。当然，如果你的系统中没有 tcp_diag 模块，ss 也可以正常运行，只是运行速度会变得稍慢。（但仍然要比 netstat 快不少。）

用数据说话

为了让你更坚决地和 netstat 说再见，我们再列举一些测试数据，以证明 ss 的快速绝非浪得虚名。

假如服务器维持着 3 万个 socket 连接，而此时系统管理员需要计算当前服务器的总连接数，则可以使用下面 3 种方法，不同方法的耗时有着明显差异：

```
netstat -at | wc  耗时 15.60 秒
ss -atr     | wc  耗时  5.40 秒（未利用tcp_diag）
ss -atr     | wc  耗时  0.47 秒（利用tcp_diag）
```

可以看到，利用了 tcp_diag 模块特性的 ss 命令，真的是飞一般的感觉。

好马配好鞍

几乎所有的 Linux 系统都会默认包含 netstat 命令，但并非所有的系统都会默认包含 ss 命令。

netstat 命令是 net-tools 软件包中的一员：

```
[root@roclinux ~]#rpm -q net-tools
net-tools-1.60-109.el6.i686
```

而 ss 命令是 iproute2 软件包中的一员：

```
[root@roclinux ~]#rpm -qf /usr/sbin/ss
iproute-2.6.32-17.el6.i686
[root@roclinux ~]#rpm -q iproute
iproute-2.6.32-17.el6.i686
```

如果你的系统无法找到 ss 命令，那么可能是没有安装 iproute2 造成的。安装的方法也非常简单：

```
[root@roclinux ~]# yum install iproute iproute-doc
```

前浪 net-tools 和后浪 iproute2

前文提到，netstat 命令是 net-tools 软件包中的一员，而 ss 命令是 iproute2 软件包中的一员。

net-tools 是一套标准的 UNIX 网络工具，用于配置网络接口、设置路由表信息、管理 ARP 表、显示和统计各类网络信息，等等，但是遗憾的是，这个工具自 2001 年起便不再更新和维护了。

长江后浪推前浪。

即将隆重登场的便是 iproute2，这是一套可以支持 IPv4/IPv6 网络的用于管理 TCP/UDP/IP 网络的软件包，这套工具由 Stephen Hemminger 负责维护和升级。

从某种意义上说，iproute2 软件包几乎可以替代掉 net-tools 软件包，具体的替代方案是如表 2 所示。

表 2　替代方案

用　　途	net-tool（被淘汰）	iproute2
地址和链路配置	ifconfig	ip addr, ip link
路由表	route	ip route
arp 表	arp	ip neigh
VLAN	vconfig	ip link
隧道	iptunnel	ip tunnel
组播	ipmaddr	ip maddr
统计	netstat	ss

ss 选项统计

ss 的选项并不算太多，除去非功能性选项（-h/-v）外，ss 共有 22 个选项。

每一个选项都是既支持短选项（如-s），也支持长选项（如--summary）。

我们不会在这里一一介绍这几个选项，因为这样既枯燥又乏味，所以，我们会从实际需求和实际问题出发，介绍最重要的选项，这样大家的学习效果会更好一些。

场景一：我想查看当前服务器的网络连接统计

```
$ ss -s
Total: 295 (kernel 312)
TCP:   48 (estab 1, closed 31, orphaned 0, synrecv 0, timewait 0/0), ports
13

Transport Total     IP        IPv6
*         312       -         -
RAW       0         0         0
UDP       2         2         0
TCP       17        12        5
INET      19        14        5
FRAG      0         0         0
```

当服务器产生大量 sockets 连接时，我们通常会使用这个命令来做宏观数据统计。

场景二：我想查看所有打开的网络端口

```
$ ss -l
Recv-Q Send-Q          Local Address:Port             Peer Address:Port
0      128                    :::webcache                   :::*
0      128                    :::http                      :::*
0      128                    :::snapenetio                    :::*
0      128                     *:snapenetio                     *:*
0      50                      *:8531                          *:*
0      9                      :::ftp                          :::*
0      9                       *:ftp                          *:*
0      128                     *:ddi-tcp-1                        *:*
0      100                   ::1:smtp                         :::*
0      100             127.0.0.1:smtp                           *:*
0      128                     *:8541                          *:*
0      128             127.0.0.1:entextxid                          *:*
0      50                      *:12421                         *:*
0      10                      *:amqp                          *:*
0      128                     *:12521                         *:*
0      50                      *:mysql                         *:*
```

如果使用-pl 参数的话，还会列出具体的程序名称，你会在输出中看到类似于下面

这样的内容：

```
("nginx",15786,6)
```

从代码中可以看到，某个 socket 连接是属于 nginx 程序的，nginx 程序的 PID 是 15786。

场景三：我想查看这台服务器上所有的 socket 连接

很简单，直接使用-a 选项即可列出所有网络连接。

```
#ss -a
```

- 如果只想查看 TCP sockets，那么就用-ta 选项，-t 表示 TCP。
- 如果只想查看 UDP sockets，那么就用-ua 选项，-u 表示 UDP。
- 如果只想查看 RAW sockets，那么就用-wa 选项，-w 表示 RAW。
- 如果只想查看 UNIX sockets，那么就用-xa 选项，-x 表示 UNIX。

11 iproute2 系列之三——iproute2 后浪推前浪

前浪——net-tools

net-tools 软件包，是 Linux 平台中非常老牌的网络工具包，它包括了如下这些很有名的命令：

1．arp：管理系统的 ARP 信息。

2．hostname：管理系统主机名。

3．ifconfig：配置网络接口。

4．netstat：展示网络连接、路由表、接口统计等信息。

5．route：管理 IP 路由。

6．ipmaddr：管理组播地址。

7．iptunnel：管理和配置隧道。

8．mii-tool：管理网络接口状态。

9．nameif：设置基于 MAC 地址的网络接口名称。

10．plipconfig：管理 PLIP 协议设备参数。

11．slattach：指定网络接口关联到特定的串行线路。

这套软件包，目前最新的发布版本是 1.6。最初是由 Phil Blundell 和 Bernd Eckenfels 来负责开发和维护的，而现在则是由 Mike Frysinger 保持着更新和维护。

在两位原作者中，Phil Blundell 很是低调，没有找到他的生活照片或工作信息；而 Bernd Eckenfels 则更加开放一些，看看他的卡通头像就能够感觉得到。

这位 Bernd 老兄，目前供职于德国的 Seeburger 公司，一家业界著名的 B2B 集成解决方案及可控文件传输（MFT）解决方案提供商，担任主任架构师一职。

长江后浪——iproute2

在前面的文章中，我们曾提到过，netstat 命令已经被 ss 命令所取代，而 ss 命令正是 iproute2 软件包中的重要一员。

iproute2 软件包，作为长江后浪，已经全面地将 net-tools 拍死在沙滩上了，包括 CentOS 在内的不少 Linux 发行版，已经在逐渐抛弃 net-tools，拥抱 iproute2 了。如果还有专家和培训机构在花大量的时间教大家 netstat、ifconfig 或者 route 命令的话，建议同学们提醒一下他们，还是抓紧时间学学 iproute2 吧，要不真的就落伍啦。

iproute2 软件包，专注在网络配置（network configuration）和流量控制（traffic control）方面，它所包括的主要工具有：

1. ip：管理路由、设备、策略和隧道等。
2. ss：展示系统套接字相关信息。
3. tc：管理流量控制策略。
4. nstat：用于网络统计。
5. bridge：管理桥接地址和设备。
6. ifcfg：进行 IP 管理，以替代 ifconfig 命令。
7. lnstat：展示网络状态。

为什么后浪可以推前浪

这是个好问题，也有同学问我，为什么 iproute2 可以取代 net-tools，成为现在主流 Linux 发行版的标配呢？我们来为大家做一个简单的说明。

很久以前，包含 arp、ifconfig 和 route 在内的 net-tools 工具集，是普遍存在于各类 Linux 发行版和 UNIX 操作系统之中的，而且大家使用起来也都挺好。但是，Linux 内核的 2.2 版本，对网络子系统进行了全面地重构，性能显著提升，全新的路由、过滤和分级策略是其他系统无法比拟的。自从 Linux 内核发布 V2.2 版本之后，net-tools 工具集就变得有些力不从心了。比如，它们无法支持 GRE 隧道技术，用户总是需要去寻找 net-tools 之外的工具才能实现一些新的网络配置策略。而 iproute2 则对 GRE 隧道等新技术支持得非常好。

到底什么是 GRE 隧道技术

这并不是本书的重点，但是大家读到这里，总是会有些好奇心，希望了解一下导致 net-tools 衰落的那个始作俑者，所以我们用一个小段落来为大家做一个科普。

首先解释一下什么是隧道技术，隧道技术是一种“网络协议的数据包被封装在另一种网络协议的数据包之中，以进行数据网络传输”的技术，这种技术其实也是 VPN 的技术基础和前提。

而 GRE 技术，则是隧道技术中应用最为普遍和广泛的一个，它的全称为通用路由封装（Generic Routing Encapsulation），是由 Net-Smiths 和 Cisco 来主导设计的。它工作在网络层，目前已被绝大多数电信设备厂商所支持。

从 net-tools 平滑过渡到 iproute2

重新回到 iproute2 的话题上，为了让大家从 net-tools 能够尽量平滑地过渡到 iproute2，我们准备了一个表格，把一些常用命令的新老对应关系展示出来，方便大家了解和记忆，如表 3 所示。

表 3 net-tools 和 iproute2 的对应关系

作　　用	net-tools 用法	iproute2 用法
展示本机所有网络接口	ifconfig	ip link show
开启某个网络接口	ifconfig eth0 up	ip link set up eth0
停止某个网络接口	ifconfig eth0 down	ip link set down eth0

续表

作　　用	net-tools 用法	iproute2 用法
给网络接口设置网络 IP 地址	ifconfig eth0 10.0.0.1/24	ip addr add 10.0.0.1/24 dev eth0
删除网络 IP 地址	ifconfig eth0 0	ip addr del 10.0.0.1/24 dev eth0
显示某个网络接口的 IP 地址	ifconfig eth0	ip addr show dev eth0
显示路由表	route -n	ip route show
添加默认网关	route add default gw 192.168.1.2 eth0	ip route add default via 192.168.1.2 eth0
删除默认网关	route del default gw 192.168.1.2 eth0	ip route replace default via 192.168.1.2 dev eth0
展示 arp 表	arp -an	ip neigh
添加 ARP 项	arp -s 192.168.1.100 00:0c:29:c0:5a:ef	ip neigh add 192.168.1.100 lladdr 00:0c:29:c0:5a:ef dev eth0
删除 ARP 项	arp -d 192.168.1.100	ip neigh del 192.168.1.100 dev eth0
查看组播信息	ipmaddr show dev eth0	ip maddr list dev eth0
展示套接字状态	netstat -l	ss -l

12 iproute2 系列之四——ip 不只是地址

ip 和它的作者

在计算机科学中，大家一提到 ip 这个词，第一个想到的大概就是 Internet Protocol Address，即 IP 地址。

在当今社会，也有很多人会想到 Intellectual Property，即知识产权。

而我们今天要讨论的既不是 IP 地址，也不是知识产权，而是 Linux 系统 iproute2 软件包中的 ip 命令，一个用来管理网络设备和路由的强大命令。

如果你阅读了本书前面的“篡权的 ss”一文，那么你应该知道 ss 命令的作者是 Alexey Kuznetosv 大牛。

而我现在要告诉大家的是，ss 命令是属于 iproute2 软件包的，整个 iproute2 软件包都是由 Alexey Kuznetosv 来主导开发的，是不是崇拜之情油然而生。

ip 命令让它们下岗

和人类的发展规律一样，每个命令也都有它的生命周期，每一个命令都将被更适应新时代的命令所取代。ip 命令的出现，也让一些老命令们退出了历史的舞台。

下面就通过表 4 来一起了解一下被 ip 取代的命令们吧。

表 4 被 ip 取代的命令

功　能	老用法	新用法
路由表	netstat -r/route	ip route
网络接口统计信息	netstat -i	ip -s link
组播	netstat -g	ip maddr
网络接口地址和链路	ifconfig	ip addr/ip link

续表

功　能	老用法	新用法
ARP	arp	ip neigh
隧道	iptunnel	ip tunnel

接下来，我们通过一些实际应用场景，学习和认识 ip 命令。

展示这台服务器的网络接口信息

以前，我们总是用 ifconfig 来查看服务器的网络接口信息，而如今，我们用 ip 命令来实现：

```
[root@roclinux ~]# ip addr show
1: lo:  mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: p1p1:  mtu 1500 qdisc mq state DOWN qlen 1000
    link/ether 00:16:d3:26:06:cc brd ff:ff:ff:ff:ff:ff
3: irda0:  mtu 2048 qdisc noop state DOWN qlen 8
    link/irda 00:00:00:00 brd ff:ff:ff:ff
4: p1p2:  mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:13:ce:86:ff:ad brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.110/24 brd 192.168.1.255 scope global p1p2
    inet6 fe80::213:ceff:fe86:ffad/64 scope link
       valid_lft forever preferred_lft forever
```

看到了吧，这台服务器上的各个网络接口的信息，完整地展示在屏幕上了，包括相应的 IP 地址、MAC 地址，等等。

当然，其中也一定包括了那个著名的回环接口（loopback），即用来连接本机的 127.0.0.1。

为网络接口添加一个 IP 地址

作为系统网络管理员，相信大家都遇到过需要为服务器网卡添加 IP 地址的任务，这是一项最基本的技能，我们用 ip 命令来实现：

```
#开始为p1p2网络接口添加IP地址
[root@roclinux ~]# ip addr add 192.168.1.111/24 dev p1p2

#看一看，已经生效拉
[root@roclinux ~]# ip addr show
1: lo:  mtu 16436 qdisc noqueue state UNKNOWN
```

```
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: p1p1:  mtu 1500 qdisc mq state DOWN qlen 1000
    link/ether 00:16:d3:26:06:cc brd ff:ff:ff:ff:ff:ff
3: irda0:  mtu 2048 qdisc noop state DOWN qlen 8
    link/irda 00:00:00:00 brd ff:ff:ff:ff
4: p1p2:  mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:13:ce:86:ff:ad brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.110/24 brd 192.168.1.255 scope global p1p2
    inet 192.168.1.111/24 scope global secondary p1p2
    inet6 fe80::213:ceff:fe86:ffad/64 scope link
       valid_lft forever preferred_lft forever
```

可以看到，我们已经为服务器的 p1p2 网络接口添加了一个新的 IP 地址 192.168.1.111，这时，我们就可以通过这个 IP 地址来访问这台服务器了：

```
localhost:~ roc$ ssh root@192.168.1.111
The authenticity of host '192.168.1.111 (192.168.1.111)' can't be
established.
RSA key fingerprint is
SHA256:nIKyIxG2WDDmBS1PpiT4T4E3hZqnJOuFuS4dvfo9SW0.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.111' (RSA) to the list of known
hosts.
root@192.168.1.111's password:
Last login: Thu Mar 17 08:06:26 2016 from 192.168.1.106
[root@roclinux ~]#
```

果然，IP 地址已生效，我们用新的 IP 地址成功登录了！

针对一个网络接口删除其 IP 地址

我们不仅可以为网络接口添加 IP 地址，也可以删除其 IP 地址，方法仍然是使用 ip addr 命令，唯一的区别就是把 add 改成 del：

```
#用del来删除IP地址
[root@roclinux ~]# ip addr del 192.168.1.111/24 dev p1p2

#查看生效情况
[root@roclinux ~]# ip addr show
1: lo:  mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: p1p1:  mtu 1500 qdisc mq state DOWN qlen 1000
    link/ether 00:16:d3:26:06:cc brd ff:ff:ff:ff:ff:ff
3: irda0:  mtu 2048 qdisc noop state DOWN qlen 8
    link/irda 00:00:00:00 brd ff:ff:ff:ff
```

```
4: p1p2:  mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:13:ce:86:ff:ad brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.110/24 brd 192.168.1.255 scope global p1p2
    inet6 fe80::213:ceff:fe86:ffad/64 scope link
       valid_lft forever preferred_lft forever
```

可以看到，刚才添加的 192.168.1.111 的 IP 地址，已经消失不见了。

激活和禁用你说了算

在系统网络运维过程中，我们有时需要将一个网卡状态置为禁用状态，以便对系统进行排错或者对网卡进行维修，这时候，就要用到 ip link 命令来控制网卡的禁用和激活了。

当然，激活和禁用网卡属于高危动作，一定要确认好之后才进行此类操作。

禁用对应的动作是 down：

```
#禁用p1p2这个网卡接口
[root@roclinux ~]# ip link set p1p2 down

#我们通过ip addr命令可以看到，p1p2网卡信息中已经出现了DOWN的字样
[root@roclinux ~]# ip addr show
1: lo:  mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: p1p1:  mtu 1500 qdisc mq state DOWN qlen 1000
    link/ether 00:16:d3:26:06:cc brd ff:ff:ff:ff:ff:ff
3: irda0:  mtu 2048 qdisc noop state DOWN qlen 8
    link/irda 00:00:00:00 brd ff:ff:ff:ff
4: p1p2:  mtu 1500 qdisc pfifo_fast state DOWN qlen 1000
    link/ether 00:13:ce:86:ff:ad brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.110/24 scope global p1p2
```

而激活则对应着 up：

```
#激活p1p2这个网卡接口
[root@roclinux ~]# ip link set p1p2 up
```

查看路由表

一说起路由表的查看与操作，你一定会想起 route 命令，熟悉的 route -n 和 route add 等。不过，这里要遗憾地告诉你，route 命令已经是 Linux 系统中明确被抛弃的命令之一，已经不再进行更新和维护了。现在流行的做法是使用 ip route。

```
#用ip route来查看系统的路由信息
[root@roclinux ~]# ip route show
192.168.1.0/24 dev p1p2  proto kernel  scope link  src 192.168.1.110
```

我要增加一条路由规则

虽然在大中型网络中，我们都是采用动态路由方式，但是静态路由仍有它的优势。

静态路由的优势可以概括为以下三个方面：

1. 由于不必交换路由信息，所以节省了一定的网络带宽。
2. 由于路由器不需要定期更新路由，从而降低了路由器的性能损耗。
3. 由于纯手工配置，因此可以一定程度上提升路由的安全性。

下面来一起看看如何手工添加和删除路由规则吧：

```
#通过add动作来增加一条路由规则
[root@roclinux ~]# ip route add 192.168.2.0/24 via 192.168.1.254

#可以看到，已经生效了
[root@roclinux ~]# ip route show
192.168.1.0/24 dev p1p2  proto kernel  scope link  src 192.168.1.110
192.168.2.0/24 via 192.168.1.254 dev p1p2

#当然可以只设定设备转发
[root@roclinux ~]# ip route add 192.168.1.5 dev p1p1

#同样生效了
[root@roclinux ~]# ip route show
192.168.1.0/24 dev p1p2  proto kernel  scope link  src 192.168.1.110
192.168.1.5 dev p1p1  scope link
192.168.2.0/24 via 192.168.1.254 dev p1p2

#删除手工配置的路由规则
[root@roclinux ~]# ip route del 192.168.1.5 dev p1p1
[root@roclinux ~]# ip route del 192.168.2.0/24 via 192.168.1.254

#好了，已经恢复如初了
[root@roclinux ~]# ip route show
192.168.1.0/24 dev p1p2  proto kernel  scope link  src 192.168.1.110
```

当然，静态路由的劣势也是显而易见的，纯手工配置对于几十台路由器的网络来说，维护成本已然是无法承受之重了。因此，应用场景也就相对狭窄，基本只在早期配置无盘网络、做网络路由实验或者小网络范围内才会使用静态路由方式。

指定默认网关

默认网关的作用很简单、很明确，就是一台主机如果找不到匹配的转发规则，那么就把数据包转发给默认指定的网关，由这个网关来处理数据包。

默认网关是不能随意指定的，一定要指定正确，否则就会造成本机网络连通方面的问题。

我们来看一下如何通过 ip route 命令指定默认网关：

```
#指定默认网关
[root@roclinux ~]# ip route add default via 192.168.1.6

#查看已生效
[root@roclinux ~]# ip route show
default via 192.168.1.6 dev p1p2
192.168.1.0/24 dev p1p2  proto kernel  scope link  src 192.168.1.110

#删除默认网关
[root@roclinux ~]# ip route del default
```

用 ip 操作 ARP 表

ARP，即 Address Resolution Protocol，用来进行 IP 地址和 MAC 地址的翻译工作。ip 命令把这部分的功能也囊括了进来，我们一起来看它的常用用法：

```
#查看本服务器的ARP表
[root@roclinux ~]# ip neigh show
192.168.1.106 dev p1p2 lladdr 98:fe:94:3d:b4:62 DELAY

#过一会儿我们再查询
[root@roclinux ~]# ip neigh show
192.168.1.106 dev p1p2 lladdr 98:fe:94:3d:b4:62 REACHABLE
```

仔细观察的话，你会发现 ARP 表的输出信息的最后一列，从 DELAY 变成了 REACHABLE，是不是很奇怪，ARP 表为什么还会动态变化呢？

原来，ARP 表的这一列表示的是邻居的可达性，它有三个可能的值，下面来为大家做一个简单介绍：

1. STALE：邻居存在，但目前处于不可达状态。
2. DELAY：探测邻居可达与否的数据包已经发出，正在等待邻居的回复。
3. REACHABLE：邻居存在，而且是可达的。

这下你应该了解为什么这一列会经常变动的原因了吧。

13 iproute2 系列之五——除了四还有六

有四了，为什么还需要六

IP 协议是整个互联网最核心的协议。而我们现在使用的主流 IP 协议（IPv4），是在 1981 年制定的，从目前互联网规模来看，已经有些力不从心了，主要问题是 IPv4 的 32 位地址空间实在有些捉襟见肘。

虽然科学家们通过 CIDR 技术、NAT 技术缓解了这一问题的爆发，但从大趋势来看，IPv4 终将有一天会耗尽而不得不退出历史的舞台。

而 IPv6，则拥有超大的地址空间，从协议设计的根本上解决了 IPv4 所面临的问题。

IPv6 小家伙呱呱落地

IPv6 这个小家伙，孕育于 1990 年，出生于 1998 年，按时间推算，到今天应该已经成年啦，不过，从实际情况来看，IPv6 协议其实才刚刚度过幼年时光，还没有真正长大呢。

实际上，IETF 早在 1990 年就提出要制定下一代的 IP（IPNG，IP Next Generation），即现在的 IPv6。而在 1998 年才真正成为因特网草案标准协议。众所周知，更换底层互联网协议并非易事，这真的需要一个非常漫长的过程。

目前，国内包括北邮、清华、中科院在内的高校，都在跟进 IPv6 技术的研究，而北京邮电大学马严教授就长期致力于国内 IPv6 技术的研究与推广工作。

IPv6 最值得记住的三个优点

首先，IPv6 拥有超级大的地址空间，达到了 128bit，而大家知道，IPv4 协议只有区区 32bit 哦。IPv6 的地址空间比 IPv4 的地址空间增大了 2^{96} 倍，夸张地说，世界上每一粒沙子都可以拥有一个属于自己的 IP 地址啦！

其次，IPv6 拥有更加灵活的协议首部。IPv6 定义了更多可选的扩展首部，增加了很多额外的功能，而且由于路由器不需要处理扩展首部，所以同时还提升了路由器的处理效率。

第三，IPv6 允许协议的二次扩充，这让 IPv6 更加与时俱进，这是 IPv4 的固定协议所无法做到的。

当然，IPv6 还有很多其他优势，例如更方便的多层级管理、更合理的协议选项、即插即用智能配置、资源预分配等。

所以说，IPv6 并不只是扩大了地址空间，而是全方位地提升了 IP 协议的可用性和可扩展性。

单播、多播和任播

在 IPv6 数据报中，目的地址可以分为三种类型，想必大家通过小标题都已经知道啦，就是单播、多播和任播。

单播，即 unicast，就是传统意义上的点到点通信。

多播，顾名思义，就是一对多通信，数据报会被传送到多台计算机。

而任播这个词，大家可能比较陌生，这是 IPv6 协议中新增的一种地址类型。任播指的是将数据报传送到一组计算机中的某一台，那到底会是哪一台呢，一般情况下会采用“就近策略”来选取。

三分钟读懂 IPv6 地址

和 IPv4 协议相比，IPv6 的地址格式是最大的变化之处。IPv6 中，每个地址都由 128bit 的二进制数字所组成，如此长的地址，到底应该如何呈现才能便于人们记忆呢？

人们首先尝试了传统的 IPv4 协议所采用的十进制点分法，但是发现，如此长的地址，实在让人们挠头，我们来体会一下：

```
113.223.141.120.255.134.255.255.0.0.18.24.150.10.255.255
```

经过多个方案的比对，最终人们还是决定采用“冒号十六进制”来呈现 IPv6 地址，这种方法会将每 16bit 的数字用一组十六进制数值来表示，其间用冒号间隔，就像这个样子：

```
45F9:8C31:FFFF:FFFF:0:1194:960B:FFFF
```

虽然看上去也不是那么易读易记，但是相比长长的十进制点分法来说，已经算是一种更好的表达了。

IPv6 地址表达中，有三个不得不提的小把戏：

（1）零省略：如果某一位是 000C，则可以直接写成 C ；

（2）零压缩：如果一个地址是 FE04:0:0:0:0:0:0:B2，那么可以直接写成 FE04::B2 ；

（3）四六混搭：在 IPv4 地址向 IPv6 地址转换时，完全可以这样写 0:0:0:0:0:0:128.10.3.2，再使用零压缩就变成了::128.10.3.2

当然这样的小把戏，也会存在使用限制，以避免滥用造成的歧义和混淆。IPv6 规定：“零压缩”把戏，在一个 IPv6 地址中只能使用一次。 至于为什么，相信你稍微思考一下就能猜到。

用 ip 命令展示和添加 IPv6 地址

ip 命令不仅可以展示我们熟悉的 IPv4 信息，还可以展示 IPv6 地址信息，就像下面这样：

```
[root@roclinux ~]#ip -6 addr show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP&gt; mtu 1500 qdisc pfifo_ fast qlen 100
inet6 fe80::210:a4ff:fee3:9566/10 scope link
inet6 2001:0db8:0:f101::1/64 scope global
inet6 fec0:0:0:f101::1/64 scope site
```

看，这台服务器的 IPv6 地址配置，已经完整地呈现出来了。

如果想为服务器手工添加一个 IPv6 地址，也并非难事，和上面介绍的 IPv4 地址添加方法异曲同工：

```
[root@roclinux ~]#ip -6 addr add 2002:0cb8:0:f311::1/64 dev eth0
```

会添加，也要会删除，聪明的你应该早已猜到，只要把 add 改为 del 即可：

```
[root@roclinux ~]# ip -6 addr del 2002:0cb8:0:f311::1/64 dev eth0
```

ip 命令深不可测

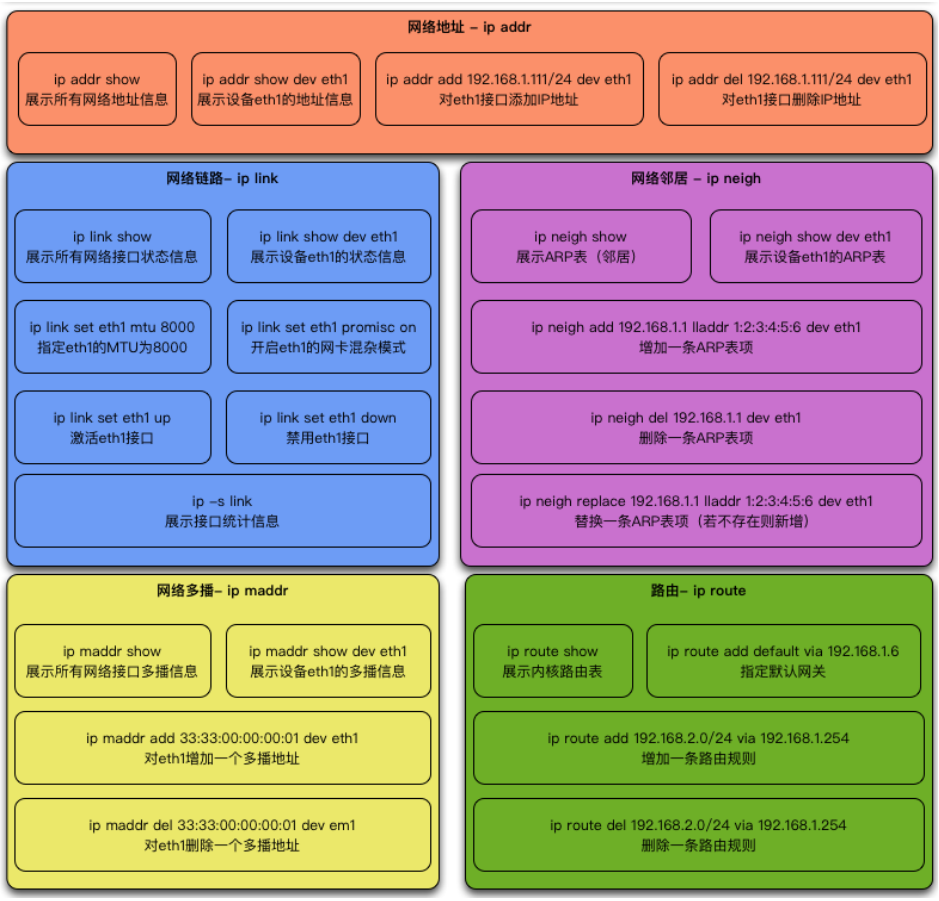
想全面地掌握和灵活地运用 ip 命令，并不是一件容易的事情，这不仅涉及命令本身的学习和使用，还涉及了计算机网络的方方面面的知识。

我们在这篇文章中并没有涉及计算机网络的所有方面，而是更多地让大家了解和

认识 ip 命令，掌握 ip 命令的最常用用法。如果将来需要深入到某一细分领域，那么就不必再从头学起了。

为了方便大家对命令的查阅和学习，我们特意对 ip 命令的各类功能进行了整理和汇总，形成了一个 ip 命令的速查表，如表 5 所示。

表 5　ip 命令速查表



14 神探 tcpdump 第一招——神探出场

神探出场

tcpdump 就好像一个神探，它有着夜视的绝技，在毫无光亮的环境中，仍然可以看到所有的东西。（好像在编美剧 HERO 一般，哈哈）

在阅读 tcpdump 系列文章之前，首先要了解和掌握下面几个知识点，否则的话，接下来的内容理解起来会略有难度。

1．了解和使用过 Linux 系统。

2．掌握网络七层协议及其作用。

3．熟悉网络协议，重点是 Ethernet/IP/TCP/UDP。

4．了解交换机、路由器所对应的协议层，并知道两者的异同点。

如果对这些知识有些遗忘了，那么强烈建议你翻出《计算机网络》教材，先好好回忆一下，然后再进入到下面的内容。

我们可以用通俗、形象、学术的表达方式来全方位地描述 tcpdump：

- 通俗地说，tcpdump 是一个抓包工具，用于抓取互联网上传输的数据包。
- 形象地说，tcpdump 就好比是国家海关，驻扎在出入境的咽喉要道，凡是要入境和出境的集装箱，海关人员总要打开箱子，看看里面都装了些啥。
- 学术地说，tcpdump 是一种嗅探器（sniffer），利用以太网的特性，通过将网卡适配器（NIC）置于混杂模式（promiscuous）来获取传输在网络中的信息包。

抓人生中的第一个包

要用 tcpdump 抓包，请记住，一定要切换到 root 账户下，因为只有 root 账户才有权限将网卡变更为适于抓包的“混杂模式”。

接下来，就是用 ifconfig 或 ip 命令查看你的服务器的网卡名称。（我的服务器的网卡叫作 eth0）

再接下来，就要开始抓人生中的第一个包啦：

```
[root@roclinux ~]# tcpdump -i eth0 -nn -X 'port 53' -c 1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
19:48:33.285838 IP 116.255.245.206.47940 > 8.8.8.8.53: 22768+ A?
www.baidu.com. (31)
0x0000: 4500 003b c341 0000 4011 3c93 74ff f5ce E..;.A..@.≤.t...
0x0010: 0808 0808 bb44 0035 0027 b457 58f0 0100 .....D.5.'.WX...
0x0020: 0001 0000 0000 0000 0377 7777 0562 6169 .........www.bai
0x0030: 6475 0363 6f6d 0000 0100 01 du.com.....
1 packets captured
1 packets received by filter
0 packets dropped by kernel
```

我们来一起解读上面这条命令：

- -i 选项：即 interface，用来指定网络接口，也就是要告诉 tcpdump 希望它去监听哪一块网卡，这在我们的服务器有多块网卡时是很有必要的。
- -nn 选项：意思是当 tcpdump 遇到协议号或端口号时，不要将这些数字转换成对应的协议名称或端口名称。比如，众所周知 21 号端口是 FTP 端口，我们希望显示“21”，而非“FTP”。
- -X 选项：告诉 tcpdump 命令，需要把协议头和包内容都原原本本地显示出来。tcpdump 会同时以 16 进制和 ASCII 的形式显示，这在进行协议分析时是绝对的利器。
- port 53：这是告诉 tcpdump 要有选择地展示所抓到的包。本例中，我们只关心源端口或目的端口是 53 的数据包，其他的数据包就不要再显示出来了。
- -c 选项：是 Count 的含义，这个选项用来设置我们希望 tcpdump 帮我们抓几个包。我设置的是 1，所以 tcpdump 只会抓一个包就收手，不会帮我再多抓哪怕一个包。

解读完命令，再来看输出的内容。从混乱的输出中，大家一定隐约地能看到 www.baidu.com，是的，我抓到的这个包，其实是用来做 DNS 解析的网络包。如果同学们对 tcpdump 输出的内容感兴趣，希望有能力读懂这些内容，那就请继续阅读下面的文章吧。

15 神探 tcpdump 第二招——两个选项

选项、过滤和输出

在第一招中，给大家演示了用 tcpdump 抓包的简单方法。接下来，我们会比较系统地来为大家介绍 tcpdump 工具了。

从我的理解来看，tcpdump 可以分为三大部分内容：第一是“选项”，第二是“过滤表达式”，第三是“输出信息”。

下面的文章，会分别从这三个方面来重点介绍。

tcpdump 的选项，总共有五十多个。在这个系列文章中，实在很难全都覆盖，所以尽量挑选重要且常用的选项来讲解， 一些不常用的选项，有兴趣的读者可以自己去了解和学习。

在《第一招》中讲到了-i 选项、-nn 选项、-c 选项和-X 选项。本节我们将继续介绍另外两个重要的选项，即-e 选项和-l 选项。

-e 选项——增加以太网帧头部信息输出

闲言少叙，直接给出例子，大家就能看到区别了：

```
#这是不带-e选项的样子
[root@roclinux ~]# tcpdump -i eth0 -c 1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
15:35:09.623964 IP leo.ssh > 111.193.194.204.55562: Flags [P.], seq
912554731:912554959, ack 1191358899, win 147, length 228
1 packets captured
5 packets received by filter
0 packets dropped by kernel

#这是带有-e选项的样子
[root@roclinux ~]# tcpdump -i eth0 -c 1 -e
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
```

```
15:36:32.516026 fa:16:3e:a4:93:99 (oui Unknown) > fa:16:3e:a8:a6:66 (oui
Unknown),  ethertype  IPv4  (0x0800),  length  282:  leo.ssh  >
111.193.194.204.55562:  Flags  [P.],  seq  912556503:912556731,  ack
1191359231, win 147, length 228
1 packets captured
4 packets received by filter
0 packets dropped by kernel
```

通过对两个命令的输出比较，我们可以看到，增加了-e 选项的命令输出中多出了MAC 地址信息，这就是以太网头里面的内容了。

为了让大家更好地理解-e 选项，我们再一起回顾一下以太网协议的头格式：

```
struct sniff_ethernet {
u_char ether_dhost[ETHER_ADDR_LEN]; /* 目的主机的地址 */
u_char ether_shost[ETHER_ADDR_LEN]; /* 源主机的地址 */
u_short ether_type; /* IP? ARP? RARP? etc */
};
```

另外，在刚才带有-e 选项的输出中，我们发现有 oui Unknown 的字样，这个 oui 是什么东西呢？在这里为大家科普一下。

OUI，即 Organizationally unique identifier，是“组织唯一标识符”，在任何一块网卡（NIC）中烧录的 6 字节 MAC 地址中，前 3 个字节都是 OUI，其代表了网卡制造商。通常情况下，该标识符是唯一的。

在我们的例子中，由于系统没有识别出其网卡制造商，所以显示了 oui Unknown 的字样。

-l 选项——让输出变为行缓冲

-l 选项的作用就是将 tcpdump 的输出变为“行缓冲”方式，这样可以确保 tcpdump 一旦遇到换行符，就会立即将缓冲的内容输出到标准输出（stdout），以便于我们利用管道或重定向方式来进行后续处理，而不会造成延迟。

众所周知，UNIX/Linux 的标准 I/O 提供了全缓冲、行缓冲和无缓冲三种缓冲方式。标准错误是不带缓冲的，而终端设备常为行缓冲，其他情况默认都是全缓冲的。

大家在使用 tcpdump 时，有时会有这样的需求：“对于 tcpdump 输出的内容，提取每一行的第一个域，即‘时间域’，并输出出来，为后续统计所用”。在这种场景下，我们就需要使用-l 选项来将默认的全缓冲变为行缓冲了。

```
[root@roclinux ~]# tcpdump -i eth0 -l |awk '{print $1}'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
```

```
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
22:56:57.571680
22:56:57.572103
22:56:57.599515
22:56:57.706286
22:56:57.888108
22:56:57.888296
22:56:57.973546
22:56:57.973768
22:56:57.975660
22:56:58.019052
22:56:58.019318
^C146 packets captured
161 packets received by filter
0 packets dropped by kernel
```

有些读者会问，如果不加-l 选项，会有什么问题么？如果不加-l 选项的话，那么只有当缓冲区全部占满时，tcpdump 才会将缓冲区中的内容输出出来，这样不仅会导致输出时总是间断不顺畅的，而且当你按下 Ctrl+C 组合键时，很可能会断到一行内容的半截部分，损坏统计数据的完整性，也影响了可读性。

好了，本文我们就专注在-e 选项和-l 选项，只要确保理解和掌握了它们两个，就足够了。

16 神探 tcpdump 第三招——选项进阶

继续选项的话题

前面说过，本书的 tcpdump 系列文章会分成“选项”、“过滤表达式”和“输出信息”三部分来讲解。

截止到目前，我们一直在围绕 tcpdump 的选项来进行讲解，已经介绍过的选项包括-i 选项、-nn 选项、-c 选项、-X 选项、-e 选项和-l 选项。

下面我们继续有关选项的话题。

-t 选项：输出时不打印时间戳

这个选项非常好理解，直接看例子就好啦：

```
#没有使用-t选项时
[root@roclinux ~]# tcpdump -i eth0 -c 1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
15:58:48.542251 IP leo.ssh > 111.193.194.204.55562: Flags [P.], seq
912766783:912767011, ack 1191362967, win 147, length 228
1 packets captured
4 packets received by filter
0 packets dropped by kernel

#使用了-t选项时
[root@roclinux ~]# tcpdump -i eth0 -c 1 -t
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
IP leo.ssh > 111.193.194.204.55562: Flags [P.], seq 912767823:912768051,
ack 1191363227, win 147, length 228
1 packets captured
3 packets received by filter
0 packets dropped by kernel
```

我们可以看到，使用了-t 选项之后，“15:58:48.542251”消失了，这就是我们抓包的时间戳信息。

至于时间为什么会精确到小数点后 6 位，这是因为 tcpdump 默认情况下就是按照微秒（microsecond）来计时的。其实，tcpdump 还支持按纳秒计时呢，开启的方法是使用--time-stamp-precision=nano 选项。

-v 选项：输出更详细的信息

加了-v 选项之后，在原有输出内容的基础之上，你还会看到 tos 值、ttl 值、ID 值、总长度、校验值等。

如果有读者对这些指标的含义感兴趣，那么建议大家可以专门去研究一下 IP 头、TCP 协议头的具体定义，就会明白它们的含义啦。

```
# 没有使用-v选项的样子
[root@roclinux ~]# tcpdump -i eth0 -c 1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
16:04:32.867226 IP leo.ssh > 111.193.194.204.55562: Flags [P.], seq
912770215:912770443, ack 1191363703, win 147, length 228
1 packets captured
4 packets received by filter
0 packets dropped by kernel

# 使用了-v选项的样子
[root@roclinux ~]# tcpdump -i eth0 -c 1 -v
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size
65535 bytes
16:04:36.498218 IP (tos 0x10, ttl 64, id 19007, offset 0, flags [DF], proto
TCP (6), length 204)
    leo.ssh > 111.193.194.204.55562: Flags [P.], cksum 0xf411 (incorrect
-> 0xb3f6), seq 912771255:912771419, ack 1191363963, win 147, length 164
1 packets captured
3 packets received by filter
0 packets dropped by kernel
```

-F 选项：指定过滤表达式所在的文件

还记得我们在第一招中所提到的命令么，我们来一起回忆一下：

```
 tcpdump -i eth0 -nn -X 'port 53' -c 1
```

这里面的‘port 53’就叫作“过滤表达式”，是用来设置我们抓包的条件的。只有满足过滤条件的网络包，才会被抓取过来。

当要设置一个非常复杂的过滤条件的时候，或者需要将一个过滤条件保存下来以便复用的时候，我们通常会把这个过滤条件存储到一个文本文件中。

在未来某天，当我们想再次使用这个过滤条件时，-F 选项就派上用场啦。请看例子：

```
#我们当初存储的过滤条件
[root@roclinux ~]# cat filter.txt
port 53

#使用-F选项来指定要读取的文件
[root@roclinux ~]# tcpdump -i eth0 -c 1 -t -F filter.txt
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
IP leo.47695 > host-192-168-0-3.domain: 41939+ A?
hss.bss.bj.baidubce.com. (41)
1 packets captured
4 packets received by filter
0 packets dropped by kernel
```

我们建立了一个 filter.txt 文本文件来存储过滤表达式，然后通过-F 来指定 filter.txt，这样 tcpdump 就会心知肚明地读取 filter.txt 中的内容作为过滤条件了。

好了，我们每篇文章求精不求多，希望大家能充分掌握。下一招，将是有关选项的最后一篇，重点讲一讲-w 和-r 两个选项。

17 神探 tcpdump 第四招——保存与回放

流量保存和回放

本节是有关“选项”知识的最后一篇啦，主要介绍-w 和-r 两个选项。tcpdump 的选项很多，有 50 多个，本系列文章没有涉及到的选项，大家可以自己通过 man tcpdump 的方式来学习。

做过网络流量分析的同学，或许都有一个共同的需求，那就是都要做“流量保存”和“流量回放”，这就恰好对应了今天要讲解的-w 选项和-r 选项。

- 流量保存就是把抓到的网络包存储到磁盘上，保存下来，为以后使用。
- 流量回放就是把历史上的某一时间段的流量，重新模拟回放出来，用于流量分析。

-w 选项：将流量保存到文件中

```
[root@roclinux ~]# tcpdump -i eth0 -w flowdata
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size
65535 bytes
^C13 packets captured
13 packets received by filter
0 packets dropped by kernel
```

通过上面的例子可以看到，通过-w 选项将流量都存储在了 flowdata 文件中了。大家是否有兴趣 less 下 flowdata，看看里面都是什么东西呢？

```
[root@roclinux ~]# less flowdata
"flowdata" may be a binary file.  See it anyway?
```

悲剧，原来是二进制格式的，无法直接通过文本方式查看。嗯，给大家卖了个关子，现在把真相告诉大家吧！

tcpdump 的-w 方式是把 raw packets（原始网络包）直接存储到文件中的，也就是存储的都是结构体数据，而非我们在屏幕上所看到的文本格式的数据，因此大家

是无法直接通过 less 命令查看的。

那怎么查看呢？大家想必也想到了，那就是用-r 选项。

-r 选项：读取 raw packets 文件

```
[root@roclinux ~]# tcpdump -r flowdata
reading from file flowdata, link-type EN10MB (Ethernet)
16:21:24.538217 IP leo.ssh > 111.193.194.204.55562: Flags [P.], seq
912799803:912799967, ack 1191371907, win 147, length 164
16:21:24.543744 IP 111.193.194.204.55562 > leo.ssh: Flags [.], ack 164,
win 4420, length 0
16:21:24.822259 IP leo.ssh > 111.193.194.204.55562: Flags [P.], seq
164:408, ack 1, win 147, length 244
16:21:25.023594 IP 111.193.194.204.55562 > leo.ssh: Flags [.], ack 408,
win 4359, length 0
16:21:28.790997 IP leo.45300 > host-192-168-0-3.domain: 55848+ A?
hss.bss.bj.baidubce.com. (41)
16:21:28.791904 IP host-192-168-0-3.domain > leo.45300: 55848 1/0/0 A
10.16.83.148 (57)
16:21:28.792338 IP leo.33269 > 10.16.83.148.ddi-tcp-1: Flags [P.], seq
4149776288:4149777312, ack 1652763622, win 2579, length 1024
16:21:28.792372 IP leo.33269 > 10.16.83.148.ddi-tcp-1: Flags [P.], seq
1024:1207, ack 1, win 2579, length 183
16:21:28.793179 IP 10.16.83.148.ddi-tcp-1 > leo.33269: Flags [.], ack 1024,
win 4877, length 0
16:21:28.793210 IP 10.16.83.148.ddi-tcp-1 > leo.33269: Flags [.], ack 1207,
win 4877, length 0
16:21:28.793527 IP 10.16.83.148.ddi-tcp-1 > leo.33269: Flags [P.], seq
1:115, ack 1207, win 4878, length 114
16:21:28.793546 IP leo.33269 > 10.16.83.148.ddi-tcp-1: Flags [.], ack 115,
win 2579, length 0
16:21:29.144843 IP 111.193.194.204.55562 > leo.ssh: Flags [P.], seq 1:53,
ack 408, win 4359, length 52
```

其实上面的命令就是在不知不觉中进行了“流量回放”，你会发现网络包被“抓”的速度都按照历史进行了回放，真像一个“时光机”啊！

由于是按 raw packets 来存储的，所以你完全可以使用-e 选项、-l 选项和过滤表达式来对输出信息进行控制，十分方便。

至此，我们有关选项的知识就探讨到这里啦。从第五招开始，我们会开始讲解过滤表达式，内容会变得更有意思了。

18 神探 tcpdump 第五招——过滤流量

师傅领进门

前四招都是围绕 tcpdump 的选项来介绍的，从这招起，我们会把目光转向更加常用的“过滤表达式”。

如果你掌握了“过滤表达式”的秘籍，那么你将具备“心无旁骛，潜心专注”的能力。

tcpdump 不仅支持指定单个“过滤表达式”来对网络包进行过滤，还支持同时接受多个过滤表达式，从这一点来说 tcpdump 还是很大肚能容的。

当你传入的过滤表达式含有 Shell 通配符时，别忘记使用单引号把表达式括起来，以防 Shell 自作主张地把含有通配符的表达式先进行了解释和通配。

如果你希望自己研究“过滤表达式”，没问题，我会告诉你如何“入门”，方法就是：

```
man pcap-filter
```

你会发现，过滤表达式大体可以分成三种过滤条件，“类型”、“方向”和“协议”，这三种条件的搭配组合就构成了我们的过滤表达式。

我只想抓 UDP 的包，不想被 TCP 的包打扰

```
[root@roclinux ~]# tcpdump -i eth0 -c 10 'udp'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
11:25:20.801612 IP 116.255.245.48.54808 > 229.111.112.12.csd-mgmt-port:
UDP, length 4
11:25:20.802120 IP 116.255.245.206.54313 > ns.sc.cninfo.net.domain:
5256+ PTR? 12.112.111.229.in-addr.arpa. (45)
11:25:21.145126 IP ns.sc.cninfo.net.domain > 116.255.245.206.54313: 5256
NXDomain 0/0/0 (45)
11:25:21.145315 IP 116.255.245.206.46658 > ns.sc.cninfo.net.domain:
15551+ PTR? 48.245.255.116.in-addr.arpa. (45)
```

```
11:25:21.153966 IP 116.255.245.43.62220 > 229.111.112.12.csd-mgmt-port:
UDP, length 4
11:25:21.180135 IP 116.255.245.61.hsrp > all-routers.mcast.net.hsrp:
HSRPv0-hello 20: state=active group=21 addr=116.255.245.33
11:25:21.231151 IP ns.sc.cninfo.net.domain > 116.255.245.206.46658:
15551 NXDomain 0/0/0 (45)
11:25:21.231430 IP 116.255.245.206.46158 > ns.sc.cninfo.net.domain:
31924+ PTR? 69.2.139.61.in-addr.arpa. (42)
11:25:21.277087 IP ns.sc.cninfo.net.domain > 116.255.245.206.46158:
31924 1/0/0 PTR ns.sc.cninfo.net. (72)
11:25:21.277824 IP 116.255.245.206.42656 > ns.sc.cninfo.net.domain: 806+
PTR? 206.245.255.116.in-addr.arpa. (46)
10 packets captured
20 packets received by filter
0 packets dropped by kernel
```

举这个例子，是为了说明 tcpdump 具有根据网络包的协议类型来进行过滤的能力，我们还可以把 udp 改为 ether、ip、ip6、arp、tcp 或 rarp 等。

或许你会问，“为什么这些协议里没有应用层协议呢？”理由其实很简单，应用层协议非基础类网络协议，经常会新增或淘汰，而且往往也没有固定的数据格式，tcpdump 更不会深入到应用层部分去智能解析。所以，你现在看到的 tcpdump 支持的协议大部分都是应用层之下的。

你我之间没有他

我想专门查看这个源机器和那个目的机器之间的网络包，不想被其他无关的网络包所打扰，该怎么做呢？

这个其实很简单，也很直观，只要设置 src（source）和 dst（destination）就好了。而且很方便的是，tcpdump 还支持使用 and 和 or 来进行搭配组合呢！

如果我们在过滤表达式中没有明确指出某个 IP 是 src 还是 dst 的话，那么默认策略是 src 或 dst 都会匹配到。

```
# 我们在本例中明确指明了dst，就表示我们希望看到网络包中的目的地址是8.8.8.8的
[root@roclinux ~]# tcpdump -i eth0 'dst 8.8.8.8'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
13:21:23.281978 IP 116.255.245.206 > google-public-dns-a.google.com:
ICMP echo request, id 23081, seq 1, length 64
13:21:24.286663 IP 116.255.245.206 > google-public-dns-a.google.com:
ICMP echo request, id 23081, seq 2, length 64
13:21:25.288612 IP 116.255.245.206 > google-public-dns-a.google.com:
ICMP echo request, id 23081, seq 3, length 64
^C
3 packets captured
```

```
5 packets received by filter
0 packets dropped by kernel
```

我们知道 8.8.8.8 是 Google 的开放 DNS，所以，从输出中可以看到，tcpdump 将其展示成了其对应的域名 google-public-dns-a.google.com。

只关注特定端口

只想看到目的端口是 53 或 80 的网络包，其他端口的我不关注。

```
[root@roclinux ~]# tcpdump -i eth0 -c 3 'dst port 53 or dst port 80'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
13:29:04.530130 IP 114.255.192.96.29832 > 116.255.245.206.http: Flags
[S], seq 3169042560, win 5840, options [mss 1460,sackOK,TS val 2949111416
ecr 0], length 0
13:29:04.530660 IP 116.255.245.206.43211 > ns.sc.cninfo.net.domain:
40188+ PTR? 206.245.255.116.in-addr.arpa. (46)
13:29:04.548589 IP 114.255.192.96.29832 > 116.255.245.206.http: Flags
[.], ack 3709396068, win 5840, options [nop,nop,TS val 2949111475 ecr
1601243970], length 0
3 packets captured
10 packets received by filter
0 packets dropped by kernel
```

我们可以设置过滤类型，上面例子中我们使用了 port 这个类型，就是来指定端口的。当然，tcpdump 还支持如下类型：

1．host：指定主机名或 IP 地址，例如‘host roclinux.cn’或‘host 202.112.18.34’。

2．net：指定网络段，例如 Src net 128.3’或‘dst net 128.3’。

3．portrange：指定端口区域，例如‘src or dst portrange 6000-6008’。

如果我们没有设置过滤类型，那么默认是 host。

19 神探 tcpdump 第六招——过滤实战

通过例子学命令

第六招，我们仍然会讲解 tcpdump 的过滤表达式。这次内容很明确，就是举例子，例子基本都来自于 man tcpdump 的内容，很直观、很受用。

【例子 1】我想抓到那些通过 eth0 网卡的，且来源是 roclinux.cn 服务器或者目标是 roclinux.cn 服务器的网络包。

```
tcpdump -i eth0 'host roclinux.cn'
```

【例子 2】我想抓那些通过 eth0 网卡的，且属于 roclinux.cn 和 baidu.com 之间通信的网络包，或者属于 roclinux.cn 和 qiyi.com 之间通信的网络包。

```
tcpdump -i eth0 'host roclinux.cn and (baidu.com or qiyi.com)'
```

【例子 3】我想获取使用 ftp 端口和 ftp 数据端口的网络包。

```
tcpdump 'port ftp or ftp-data'
```

大家是不是会有一个疑问“这个 ftp/ftp-data 到底对应哪个端口？除了 ftp/ftp-data 外，还有哪些服务名称我可以直接用呢？”

这是个好问题，答案现在揭晓。

在 Linux 系统中，/etc/services 这个文件里面，存储着所有知名服务和传输层端口的对应关系。这个对应关系是由 IANA 组织（the Internet Assigned Numbers Authority，互联网数字分配机构）来全权负责的，你可以到这个链接 http://www.iana.org/assignments/port-numbers 查询。

如果直接把/etc/services 里的 ftp 对应的端口值从 21 改为了 8888，那么 tcpdump 就会去抓端口含有 8888 的网络包了。

【例子 4】我想获取 roclinux.cn 和 baidu.com 之间建立 TCP 三次握手中的第一个网络包，即带有 SYN 标记位的网络包，另外，目的主机不能是 qiyi.com。

```
tcpdump 'tcp[tcpflags] & tcp-syn != 0 and not dst host qiyi.com'
```

这个语句比较复杂，想把这个命令解读清楚也的确不容易，需要你具备计算机网络专业知识才行，尤其是三次握手的原理。如果大家理解起来比较吃力，那可要去复习计算机网络知识了。

【例子 5】打印 IP 包长度超过 576 字节的网络包。

```
tcpdump 'ip[2:2] > 576'
```

【例子 6】打印广播包或组播包，同时数据链路层不是通过以太网媒介进行的。

```
tcpdump 'ether[0] & 1 = 0 and ip[16] >= 224'
```

最后三个例子，或许你看得有些晕头转向，尤其是 ip[2:2]和 ip[16]这样的内容。

没关系，这三个例子是为了让大家先有一个感官认识，在接下来的文章中会为大家答疑解惑。

20 神探 tcpdump 第七招——过滤高手

磨刀不误砍柴工

在上一招中，最后的三个例子比较复杂，大家可能会感觉云里雾里。待第七招读完，保证大家能读懂它们！

```
tcpdump 'tcp[tcpflags] & tcp-syn != 0 and not dst host qiyi.com'

tcpdump 'ip[2:2] > 576'

tcpdump 'ether[0] & 1 = 0 and ip[16] >= 224'
```

在阅读本文前，希望读者们先复习一下 ETHER/IP/TCP/UDP 等协议的包头定义格式，最好能知道每一位（bit）的作用和含义。

磨刀不误砍柴工，如果本系列的前六篇文章你都仔细看过了，同时，几种协议的包格式也都了然于心了，那么我们就切入正题，介绍下过滤表达式的高级语法。

理解[2:2]这种语法格式

其实我们需要着重讲解的就是这种语法格式，即 proto [expr : size]，只要掌握了这种语法格式，相信大家就能看懂上面的三个稀奇古怪的表达式了。

proto 就是 protocol 的缩写，表示这里要指定的是某种协议的名称，比如 ip、tcp、ether。

proto 这个位置，我们可以指定的协议类型大体包括：

- ether：链路层协议。
- fddi：链路层协议。
- tr：链路层协议。
- wlan：链路层协议。
- ppp：链路层协议。
- slip：链路层协议。

- link：链路层协议。
- ip：网络层。
- arp：网络层。
- rarp：网络层。
- tcp：传输层。
- udp：传输层。
- icmp：网络层。
- ip6：网络层。

proto [expr : size]中，expr 用来指定数据报偏移量，表示从某个协议的数据报的第多少位开始提取内容，默认的起始位置是 0。而 size 表示从偏移量的位置开始提取多少个字节，比如大家看到的 1、2、4。

如果只设置了 expr，而没有设置 size，则默认提取 1 个字节。比如 ip[2:2]，就表示提取出第 3、4 个字节；而 ip[0]则表示提取 ip 协议头的第 1 个字节。

在提取了特定内容之后，我们就需要设置我们的过滤条件了。我们可用的“比较操作符”包括：>、<、>=、<=、=和!=，共 6 个。

掌握了上面内容之后，我可以很负责任地告诉你，你已经掌握了 tcpdump 过滤表达式的最重要的语法知识了。我们先来小试牛刀，看一个示例：

```
ip[0] & 0xf != 5
```

要想理解这个示例，那就一定要对 IP 协议头有充分的认识和掌握，所以，我们贴出了 IP 协议头的示意图，好让大家得到充分的复习，如图 6 所示。

如果大家忘记了其中的字段的含义和作用，那可一定要去复习一下《计算机网络》。

如果大家是已经做足了功课才来阅读本文的，那么你一定知道，IP 协议的第 1-4 位，表示 IP 版本号，其值可能是 0100（即 IPv4）或者 0110（即 IPv6）。第 5-8 位表示首部长度，其值的单位是“4 字节”。如果首部长度为默认的 20 字节的话，那么此值就应该为 5。如果大于 5，那么表示此 IP 包中包含了可选字段。

注意，“IP 版本号”和“首部长度”合在一起，正好占一个字节。而 ip[0]，表示提取 IP 头从第 0 字节开始的 1 个字节的内容，也就是提取“IP 版本号”和“首部长度”这两个域的合体。

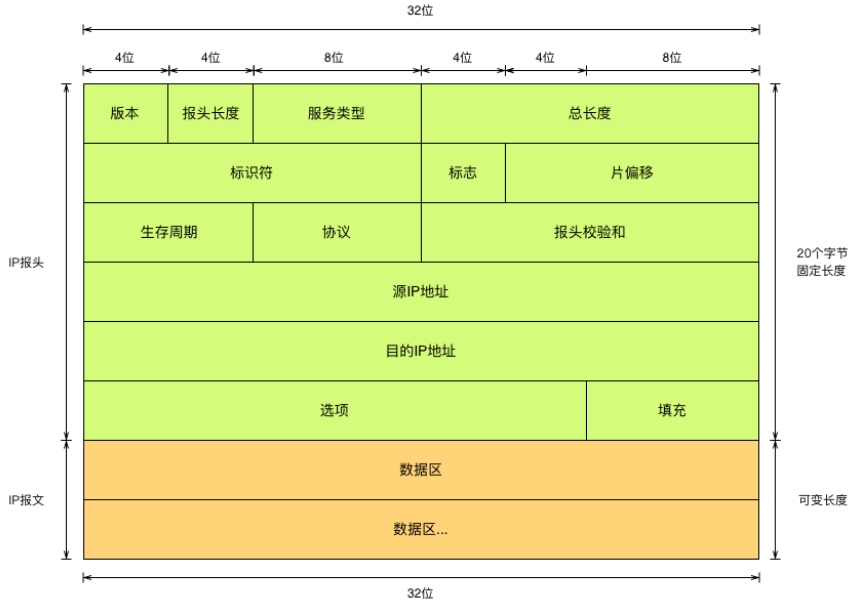


图 6　IP 协议头

而 0xf 中的 0x 表示十六进制，f 是十六进制数，转换成 8 位的二进制数是“0000 1111”。可见，“ip[0] & 0xf”提取的就是 IP 的“首部长度”字段的值。

有了上面这些分析，大家应该可以很清楚地知道，“ip[0] & 0xf != 5”的含义是：如果首部长度不等于 5，就满足了过滤条件。也就是说，要求 IP 包的首部中含有可选字段。

一些可读的名称

大家可能已经有所体会，在写过滤表达式时，你需要把协议格式完全背在脑子里，才能把表达式写对。但这样的确有些反人类。

为了让 tcpdump 工具更人性化一些，一些常用的字段，可以通过一些名称来代替。比如 TCP 协议头中的六个 tcpflags 标志字段域，如图 7 所示。

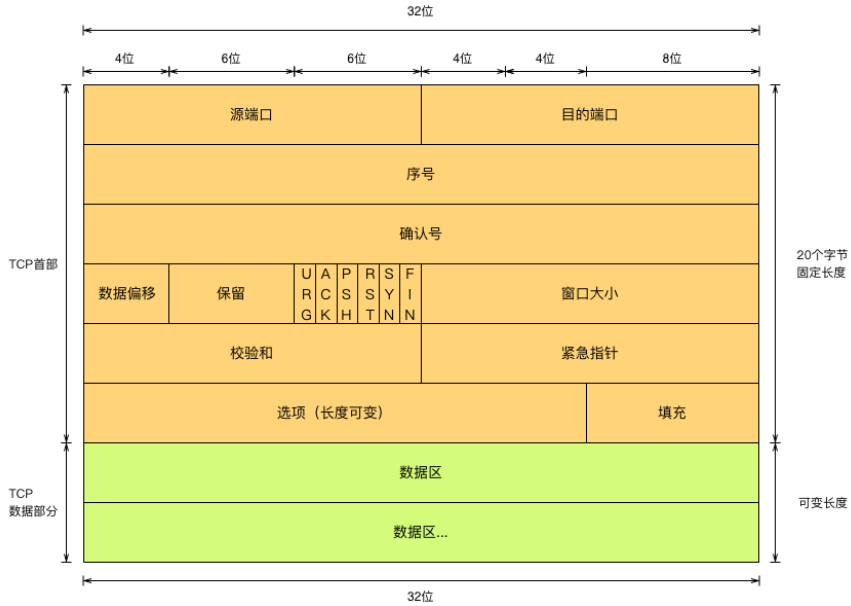


图 7　TCP 协议头

用可读的变量表示就是：tcp-urg、tcp-ack、tcp-push、tcp-rst、tcp-syn 和 tcp-fin。

最后，再补充一个知识点，那就是，如果一个过滤表达式中包含多个过滤条件，那么我们可以使用逻辑关系符将它们串联起来，其中：

- !或 not：表示“否定”。
- &&与 and：表示“与”。
- ||与 or：表示“或”。

好了，大功告成，如果你仔细阅读并掌握了上面的内容，那么我相信你有能力自己来分析下面这三个语句，而且一定能明白它们的“言下之意”的！

```
tcpdump 'tcp[tcpflags] & tcp-syn != 0 and not dst host qiyi.com'
tcpdump 'ip[2:2] > 576'
tcpdump 'ether[0] & 1 = 0 and ip[16] >= 224'
```

21 神探 tcpdump 第八招——输出解读

终于讲到输出内容了

在这个系列文章的开篇部分，曾提到“学习 tcpdump，要掌握的是三个部分，即选项、过滤表达式和输出内容”。本篇文章，我们就进入“输出内容”的讲解。

还记得我们在第一篇文章里的“抓人生中的第一个包”么，现在我们就来好好剖析剖析这人生中的第一个包。

```
[root@roclinux ~]# tcpdump -i eth0 -nn -X 'port 53' -c 1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
19:48:33.285838  IP  116.255.245.206.47940  >  8.8.8.8.53:  22768+  A?
www.baidu.com. (31)
0x0000: 4500 003b c341 0000 4011 3c93 74ff f5ce E..;.A..@.&lt;.t...
0x0010: 0808 0808 bb44 0035 0027 b457 58f0 0100 .....D.5.'.WX...
0x0020: 0001 0000 0000 0000 0377 7777 0562 6169 .........www.bai
0x0030: 6475 0363 6f6d 0000 0100 01 du.com.....
1 packets captured
1 packets received by filter
0 packets dropped by kernel
```

解释第 2 行

```
"tcpdump: verbose output suppressed, use -v or -vv for full protocol
decode"
```

这一行，是一句贴心的提醒，简单易懂，就是说你的命令里没有用到-v 和-vv，如果希望看到更全的输出内容，那么可以使用这两个选项。如果你有兴趣，可以试试，看看加上-vv 后，到底输出内容里会多些什么。

解释第 3 行

```
"listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes"
```

这一句表示我们监听的是流经 eth0 这个网卡的网络包，且它的链路层是基于以太

网的，要抓的包大小限制是 65535 字节。

包大小限制值可以通过-s 选项来设置，如果你追求高性能，那么建议把这个值调低，这样可以有效避免在大流量情况下的丢包现象。

解释第 4 行

```
"19:48:33.285838 IP 116.255.245.206.47940 > 8.8.8.8.53: 22768+ A?
www.baidu.com. (31)"
```

“19:48:33.285838”，分别对应着这个包被抓到的“时”、“分”、“秒”、“微秒”。

“IP”，表示这个包在网络层，是 IP 包。

“116.255.245.206.47940”，表示这个包的源 IP 为 116.255.245.206，源端口为 47940。

“>”，表示数据包的传输方向。

“8.8.8.8.53”，表示这个包要发向的目的端 IP 是 8.8.8.8，目标端口为 53，也就是我们熟知的 DNS 服务端口。

“22768+ A? www.baidu.com. (31)”，这是 DNS 协议的内容，即请求 www.baidu.com 的 A 记录。

解释第 5/6/7/8 行

```
0x0000: 4500 003b c341 0000 4011 3c93 74ff f5ce E..;.A..@.<.t...
0x0010: 0808 0808 bb44 0035 0027 b457 58f0 0100 .....D.5.'.WX...
0x0020: 0001 0000 0000 0000 0377 7777 0562 6169 .........www.bai
0x0030: 6475 0363 6f6d 0000 0100 01           du.com.....
```

这几行便是 IP 包的内容了，是除去了以太网之后剩下的内容。其中，左侧部分是十六进制内容，右侧部分是相应的 ASCII 码内容。如果想看懂上面这些十六进制数字，则需要大家对 IP、TCP/UDP 的包格式很熟悉才可以。

下面我们就来给大家解读下那些“晦涩的十六进制数字”。

```
0x0000: 【45】00 003b c341 0000 4011 3c93 74ff f5ce E..;.A..@.<.t...
0x0010: 0808 0808 bb44 0035 0027 b457 58f0 0100 .....D.5.'.WX...
0x0020: 0001 0000 0000 0000 0377 7777 0562 6169 .........www.bai
0x0030: 6475 0363 6f6d 0000 0100 01           du.com.....
```

如上，最外层是 IP 数据包，最开始的一个字节（8 位）中，前 4 位（bit）表示 IP 的版本，此处为 4，表示这是一个 IPv4 版本的 IP 包。后 4 位（bit）表示此 IP 包

的首部长度，此处的数字是 5，由于单位是“4 字节”，因此可以计算得出这个 IP 包的首部长度是固定的 20 字节。如下（括号）部分都是 IP 数据包的首部部分：

```
0x0000: 【4500 003b c341 0000 4011 3c93 74ff f5ce E..;.A..@.<.t...
0x0010: 0808 0808】 bb44 0035 0027 b457 58f0 0100 .....D.5.'.WX...
0x0020: 0001 0000 0000 0000 0377 7777 0562 6169 .........www.bai
0x0030: 6475 0363 6f6d 0000 0100 01           du.com.....
```

如下所示，在 IP 版本和首部长度之后，接下来的一个字节（8 位）是“00”，这是 IP 协议的服务类型域（TOS），由于已经很少使用，因此此处被置为 00。

```
0x0000: 45【00】003b c341 0000 4011 3c93 74ff f5ce E..;.A..@.<.t...
0x0010: 0808 0808 bb44 0035 0027 b457 58f0 0100 .....D.5.'.WX...
0x0020: 0001 0000 0000 0000 0377 7777 0562 6169 .........www.bai
0x0030: 6475 0363 6f6d 0000 0100 01           du.com.....
```

如下所示，后面的 2 字节（16 位）是“003b”，表示整个 IP 包的总长度（首部长度+数据长度），单位是字节。因此可以知道这个 IP 包的总长度是 59 字节（0x3b 需要转换为十进制）。

```
0x0000: 4500 【003b】 c341 0000 4011 3c93 74ff f5ce E..;.A..@.<.t...
0x0010: 0808 0808 bb44 0035 0027 b457 58f0 0100 .....D.5.'.WX...
0x0020: 0001 0000 0000 0000 0377 7777 0562 6169 .........www.bai
0x0030: 6475 0363 6f6d 0000 0100 01           du.com.....
```

再向下的 2 字节（16 位）是“标识域”，如果 IP 包的大小超过了数据链路层的 MTU 限制，就需要对 IP 包进行分拆，此时就要用这个域来表示哪些包在分拆前是同一组的。此处的标识域值为 0xc341。

```
0x0000: 4500 003b 【c341】 0000 4011 3c93 74ff f5ce E..;.A..@.<.t...
0x0010: 0808 0808 bb44 0035 0027 b457 58f0 0100 .....D.5.'.WX...
0x0020: 0001 0000 0000 0000 0377 7777 0562 6169 .........www.bai
0x0030: 6475 0363 6f6d 0000 0100 01           du.com.....
```

再继续向下看，便是 3 位（bit）的标志位，最高位为保留位，中间一位为 DF（don’t fragment），最低位为 MF（more fragments）。可以看到，这三位是用来控制 IP 拆包后的组装所用。由于此包没有拆包，因此这三位都被置为 0，如下所示。

```
0x0000: 4500 003b c341 【0】000 4011 3c93 74ff f5ce E..;.A..@.<.t...
0x0010: 0808 0808 bb44 0035 0027 b457 58f0 0100 .....D.5.'.WX...
0x0020: 0001 0000 0000 0000 0377 7777 0562 6169 .........www.bai
0x0030: 6475 0363 6f6d 0000 0100 01           du.com.....
```

和上面的标志位配合的是紧接着的 13 位（bit）的片偏移，但由于本包没有拆包，因此此域也无用，故而标记为 0。

```
0x0000: 4500 003b c341 【0000】 4011 3c93 74ff f5ce E..;.A..@.<.t...
0x0010: 0808 0808 bb44 0035 0027 b457 58f0 0100 .....D.5.'.WX...
0x0020: 0001 0000 0000 0000 0377 7777 0562 6169 .........www.bai
0x0030: 6475 0363 6f6d 0000 0100 01           du.com.....
```

如下，紧接着是 8 位（bit）的 TTL（Time To Live，即生存周期），此包的值为 0x40，换算成十进制是 64。这表明这个网络包，如果经过了超过 64 个中间路由节点，则认为目的地不可达，中间路由器会将此包抛掉。

```
0x0000: 4500 003b c341 0000 【40】11 3c93 74ff f5ce E..;.A..@.<.t...
0x0010: 0808 0808 bb44 0035 0027 b457 58f0 0100 .....D.5.'.WX...
0x0020: 0001 0000 0000 0000 0377 7777 0562 6169 .........www.bai
0x0030: 6475 0363 6f6d 0000 0100 01           du.com.....
```

接下来的 8 位（bit）为协议域，用于指代上一层协议类型。此处的值为 0x11，对应十进制的 17，而 17 是 UDP 协议的代号，因此可以知道这个网络包所用的传输层协议是 UDP，而非 TCP（TCP 的协议号是 6、TCMP 的协议号是 1）。

```
0x0000: 4500 003b c341 0000 40【11】 3c93 74ff f5ce E..;.A..@.<.t...
0x0010: 0808 0808 bb44 0035 0027 b457 58f0 0100 .....D.5.'.WX...
0x0020: 0001 0000 0000 0000 0377 7777 0562 6169 .........www.bai
0x0030: 6475 0363 6f6d 0000 0100 01           du.com.....
```

接下来的 2 个字节表示 IP 首部校验和，此处计算出来的结果是 3c93。

```
0x0000: 4500 003b c341 0000 4011 【3c93】 74ff f5ce E..;.A..@.<.t...
0x0010: 0808 0808 bb44 0035 0027 b457 58f0 0100 .....D.5.'.WX...
0x0020: 0001 0000 0000 0000 0377 7777 0562 6169 .........www.bai
0x0030: 6475 0363 6f6d 0000 0100 01           du.com.....
```

再往下就是大家非常熟悉的 4 字节的 IP 源地址，即“74 ff f5 ce”，转换成 IP 地址则为 116.255.245.206。

```
0x0000: 4500 003b c341 0000 4011 3c93 【74ff f5ce】 E..;.A..@.<.t...
0x0010: 0808 0808 bb44 0035 0027 b457 58f0 0100 .....D.5.'.WX...
0x0020: 0001 0000 0000 0000 0377 7777 0562 6169 .........www.bai
0x0030: 6475 0363 6f6d 0000 0100 01           du.com.....
```

以此类推，再下面的 4 字节则为 IP 目的地址，即“08 08 08 08”，转换成 IP 地址则为 8.8.8.8，即著名的 Google-DNS 服务器地址。

```
0x0000: 4500 003b c341 0000 4011 3c93 74ff f5ce E..;.A..@.<.t...
0x0010: 【0808 0808】 bb44 0035 0027 b457 58f0 0100 .....D.5.'.WX...
0x0020: 0001 0000 0000 0000 0377 7777 0562 6169 .........www.bai
0x0030: 6475 0363 6f6d 0000 0100 01           du.com.....
```

至此，网络层 IP 协议的首部 20 字节已经分析完了。再向下我们即将进入传输层 UDP 协议的包分析阶段。

UDP 协议包分析

相对于 TCP 包来说，UDP 包的首部还是比较简单的，总共只有 8 个字节，如下括号部分便是 UDP 首部部分：

```
0x0000: 4500 003b c341 0000 4011 3c93 74ff f5ce E..;.A..@.<.t...
0x0010: 0808 0808 【bb44 0035 0027 b457】 58f0 0100 .....D.5.'.WX...
0x0020: 0001 0000 0000 0000 0377 7777 0562 6169 .........www.bai
0x0030: 6475 0363 6f6d 0000 0100 01           du.com.....
```

UDP 首部的前 2 个字节为源端口，此处为“bb 44”，即 47940。而接下来的 2 字节为目的端口，值为“00 35”，即 53（DNS 服务）。

```
0x0000: 4500 003b c341 0000 4011 3c93 74ff f5ce E..;.A..@.<.t...
0x0010: 0808 0808 【bb44】【0035】 0027 b457 58f0 0100 .....D.5.'.WX...
0x0020: 0001 0000 0000 0000 0377 7777 0562 6169 .........www.bai
0x0030: 6475 0363 6f6d 0000 0100 01           du.com.....
```

而接下来的 2 字节则表示 UDP 包的总长度（报头+数据部分）。此处的值为“00 27”，转换成十进制则为 39，表示此 UDP 包的总长度为 39 字节，减去首部的 8 字节外，还有 31 字节来存储真正要传输的数据。

```
0x0000: 4500 003b c341 0000 4011 3c93 74ff f5ce E..;.A..@.<.t...
0x0010: 0808 0808 bb44 0035 【0027】【b457】 58f0 0100 .....D.5.'.WX...
0x0020: 0001 0000 0000 0000 0377 7777 0562 6169 .........www.bai
0x0030: 6475 0363 6f6d 0000 0100 01           du.com.....
```

而 UDP 首部的最后两个字节则是“校验和”部分，此处的值为 0xb457。

再向下的部分，则是应用层协议的内容（本例中是 DNS 协议）。如果读者有兴趣，可以继续了解下 DNS 协议、HTTP 协议、FTP 协议等众多的应用层协议，这非常有利于大家在学习协议、追查网络问题时，透过现象看本质。

```
0x0000: 4500 003b c341 0000 4011 3c93 74ff f5ce E..;.A..@.<.t...
0x0010: 0808 0808 bb44 0035 0027 b457 【58f0 0100 .....D.5.'.WX...
0x0020: 0001 0000 0000 0000 0377 7777 0562 6169 .........www.bai
0x0030: 6475 0363 6f6d 0000 0100 01】            du.com.....
```

好了，通过本篇文章，大家应该了解了 TCP 包、UDP 包的解读方法，是不是已经有冲动去自己抓一个包来解读解读了呢。

22 神探 tcpdump 终结招——七个秘籍

如果你已经认真阅读过了前面的八篇文章，那么 tcpdump 的最主要招数你应该已经驾轻就熟了，恭喜你。

在最后的“终结招”中，我们会给大家介绍一些之前没有提到的“小秘籍”，让大家在追查网络问题、进行协议分析时，可以用得上。

【秘籍一】

使用-A 选项，则 tcpdump 只会显示 ASCII 形式的数据包内容，而不会再以十六进制形式显示。

【秘籍二】

使用-XX 选项，则 tcpdump 会从以太网部分就开始显示网络包内容，而不是仅从网络层协议开始显示。

【秘籍三】

使用如下命令，则 tcpdump 会列出所有可以选择的抓包对象。

```
# tcpdump -D
1.eth0
2.any (Pseudo-device that captures on all interfaces)
3.lo
```

【秘籍四】

如果想查看哪些 ICMP 包中“目标不可达、主机不可达”的包，请使用下面这样的过滤表达式：

```
icmp[0:2]==0x0301
```

【秘籍五】

要想提取 TCP 协议的 SYN、ACK 和 FIN 标识字段，则语法如下。

```
tcp[tcpflags] & tcp-syn
```

```
tcp[tcpflags] & tcp-ack
tcp[tcpflags] & tcp-fin
```

【秘籍六】

要想提取 TCP 协议里的 SYN-ACK 数据包，则不但可以使用上面的方法，还可以直接使用最本质的方法。

```
tcp[13]==18
```

【秘籍七】

如果要抓取一个区间内的端口，则可以使用 portrange 语法：

```
tcpdump -i eth0 -nn 'portrange 52-55' -c 1  -XX
```

好了，我们的“神探 tcpdump”系列文章至此结束啦。我希望通过这一系列文章，让大家可以不再惧怕 tcpdump，不再害怕抓包和分析包，这就足够了。

23 nc，一只可爱的网猫

网猫出场

命令 nc，全名 netcat，中文叫作“网猫”，nc 的 man 中提到：

```
The nc (or netcat) utility is used for just about anything under the sun
involving TCP or UDP.
```

这句话翻译成中文就是：“nc 工具能胜任全天下的跟 TCP/UDP 相关的一切操作。”

这种推崇备至、不遗余力地赞赏在 Linux 的 man 手册中极少见到。nc 确实很能干，它可以打开 TCP 链接、发送 UDP 包、监听 TCP/UDP 端口、进行端口扫描，是网络世界里的一只既可爱又靠谱的网猫。

你有 QQ 我有网猫

当我第一次使用 QQ 时，我觉得它是世界上最伟大的发明，居然能够通过软件和远端好友聊天，真是太神奇了！后来接触了 nc，才知道原来 QQ 的老祖宗早已问世许久，崇拜之情更是油然而生。

有人会说现在的聊天工具多如牛毛，又有谁会去使用 nc 提供的聊天功能呢？

其实现在很多企业、实验室出于安全等方面的考虑，对第三方软件的安装和外网的访问有着非常严格的限制，想要跟外界哪怕是坐在隔壁的朋友聊天，都是一件不容易的事情。而 nc 却能帮你搞定这个问题，下面我们就来看看如何使用 nc 来建立客户端—服务器的连接，从而使两端能够进行文字沟通。

在本例中我们扮演服务器的角色。在本地打开一个终端，使用-l 选项监听 12345 端口，等待其他终端的连接。执行完命令后屏幕没有任何反应。

```
[roc@roclinux ~]$ nc -l 12345
```

然后，我们在另一台服务器上尝试发起连接：

```
[roc@roclinux ~]$ nc 116.255.245.207 12345
```

我们指定了 IP 和端口，来连接我们的服务器端。命令执行完毕之后，此时客户端和服务器端的连接就建立起来了。连接一旦建立，在任何一端输入的文字都将出现在另一端，在任何一端执行组合键 Ctrl+D（即 EOF）都将断开连接。

有兴趣的同学可以自己去尝试一下这种原始的“聊天软件”。

端口扫描

黑客往往会通过端口扫描来找到系统开放了哪些端口，并从中找到服务漏洞，继而实施系统入侵。黑客聪明，管理员需要更聪明，因此系统管理员也需要进行端口扫描，以便及早发现系统的潜在漏洞并进行修复。虽然 nmap 在端口扫描领域是绝对的优选工具，但如果我们只需要探测端口的开放状态，那么 nc 也完全可以胜任。

我们来实际实施一次端口扫描：

```
[roc@roclinux ~]$ nc -z -v -n -w 2 127.0.0.1 20-23
nc: connect to 127.0.0.1 port 20 (tcp) failed: Connection refused
nc: connect to 127.0.0.1 port 21 (tcp) failed: Connection refused
Connection to 127.0.0.1 22 port [tcp/*] succeeded!
nc: connect to 127.0.0.1 port 23 (tcp) failed: Connection refused
```

上面命令扫描本地 20-23 端口，发现 22 端口能够连接成功，也就是 22 号端口此时处于开放状态。下面解释该命令涉及的几个选项：

- -z 选项：一旦建立连接后马上断开，而不发送和接收任何数据。
- -v 选项：打印详细输出信息。
- -n 选项：直接使用 IP 地址，而不使用域名服务器来查询其域名。
- -w 选项：设置连接的超时时间，单位为秒。
- -u 选项：使用 UDP 建立连接。上面命令无此设置，则表示使用 TCP 建立连接。

当连接到服务器上的某个端口时，监听该端口的服务会传送一个 Banner 信息（横幅广告一般的欢迎信息），这个 Banner 信息一般都会说明自身程序的版本号等信息。黑客们就可以利用这些信息做进一步地分析并制定入侵策略。

从上面的例子我们可以发现，22 号端口处于开放状态，于是，我们尝试用 nc 来获取 22 号端口的 Banner 信息。

```
[roc@roclinux ~]$ nc -v 127.0.0.1 22
```

```
Connection to 127.0.0.1 22 port [tcp/ssh] succeeded!
SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2
```

从输出的内容我们可以看出，其版本信息为 SSH-2.0-OpenSSH_6.6.1p1。如果这一版本的 SSH 有什么漏洞的话，那么黑客就可以利用这些漏洞来入侵我们的系统了，江湖又将掀起一场血雨腥风。

传输文件

说起在机器间传输文件，大家都会微微一笑。这么简单的工作，ftp 和 scp 早就可以轻松搞定了，为什么还需要 nc 呢？

其实用 nc 来传输文件有着独特的优势，即只需选择一个端口就可以在两台机器之间进行文件传输，而不需要进行任何配置，即不像 ftp 和 scp 在登录阶段还需要验证用户名、密码这样的安全信息。

下面，我们就来演示在两台服务器间进行的文件传输活动。

在服务器端，也就是文件的发送端，我们启动监听端口，准备好文件，等待客户端来接收：

```
[roc@roclinux ~]$ nc -v -l 12345 < book_out.txt
```

在客户端，也就是文件的接收端，会使用如下命令来接收服务器端的数据，并重定向到文件中：

```
[root@roclinux ~]# nc -v -n 116.255.245.207 12345 > book_in.txt
[root@roclinux ~]# ls -hl book_in.txt
-rwxr--r-- 1 root root 6.4K Apr 13 18:26 book_in.txt
```

此时，再回到服务器端，可以看到这样的信息输出：

```
[roc@roclinux ~]$ nc -v -l 12345 < book_out.txt
Connection from 180.76.189.113 port 12345 [tcp/italk] accepted
```

这说明客户端已经来取数据了。一旦数据传输完毕，连接会自动断开。

服务器端，也就是使用-l 选项进行端口监听的一端，不一定非要是文件的发送端，完全可以反过来作为文件的接收端。

```
[roc@roclinux ~]$  nc -v -l 12345 > book_in.txt
```

此时的客户端，则充当文件发送端的角色。

```
[root@roclinux ~]# nc -n 116.255.245.207 12345 ≤ book_out.txt
```

瞬间剧情反转了，是不是很有意思呢？

传输文件夹

如果我们需要传送多个文件，甚至是文件夹，是不是就需要使用 ftp 或者 scp 了呢？

其实 nc 和 tar 强强联手，就完全可以搞定这个需求！我们来看一下如何实现：

```
#服务器端，将tar的输出通过管道传给nc
[roc@roclinux ~]$ tar -cvPf - /root/book/ | nc -l 12345
/root/book/
/root/book/chapter1.txt
/root/book/chapter2.txt

#在客户端连接12345端口，然后将接收到的数据通过tar直接解包
[root@roclinux ~]# nc -n 116.255.245.207 12345 | tar -xvPf -
/root/book/
/root/book/chapter1.txt
/root/book/chapter2.txt
```

为了节省带宽，我们也可以将文件夹打成压缩包再进行传送，下面的命令是使用 gzip 压缩和解压缩。如果要使用 bzip2，则将 z 选项改成 j 选项即可。

- 服务器端：tar -czvPf - /root/book/ | nc -l 12345
- 客户端：nc -n 116.255.245.207 12345 | tar -xzvPf -

好了，这只网猫是不是既可爱又靠谱呢。以后在文件传输、简单信息传输、端口扫描等场景下，可不要忘记它哦！

24 ssh-copy-id，帮你建立信任

不堪回首的手工时代

对于做运维的同学来说，给两台 UNIX/Linux 机器建立 ssh 信任关系是再平常不过的事情了。

不知道大家之前建立信任关系是采用什么方法呢，反正我是纯手工创建。

如果需要“machineA 机器的 nameA 账号”建立到“machineB 机器的 nameB 账号”的 ssh 信任关系，达到无须输入密码即可登录的目的，那么我一般是这样做的：

1．将 machineA 机器的/home/nameA/.ssh/id_rsa.pub 文件的内容复制出来。

2．登录到 machineB 机器的/home/nameB/.ssh 中，如果不存在则创建 authorized_keys 文件，将第 1 步中的内容追加到文件尾部。

3．检查 authorized_keys 文件的权限，确保其 group/other 位没有 w 权限。

4．登录到 machineA 机器，测试 ssh 信任关系是否建好。

其实上面的添加机器信任关系的方法很不友好，需要全手工操作，而且要两台机器之间来来回回切换，且操作正确性完全由人控制，很容易出现问题和错误。

现在，隆重推出“SSH 信任关系自动化建立工具”：ssh-copy-id。这是一个划时代的时刻，让人类学会了使用工具，哈哈。

五分钟学会 ssh-copy-id

在不建立 ssh 信任关系的情况下，从 machineA 机器的 nameA 登录到 machineB 机器的 nameB，可以看出是需要输入密码的：

```
[nameA@machineA]$ ssh nameB@machineB -p 22000
nameB@machineB's password:
```

我们现在就用新命令来建立信任关系：

```
[nameA@machineA]$ ssh-copy-id nameB@machineB
/usr/bin/ssh-copy-id: ERROR: No identities found
```

却提示“No identities found”，即“没有找到标识”。这是因为我们的 nameA 账号还没有自己的公钥私钥。

我们需要先为 nameA 账号建立自己的公钥私钥，建立好之后，会在 /home/nameA/.ssh 里会多出 id_rsa（私钥）和 id_rsa.pub（公钥）两个文件：

```
[nameA@machineA]$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/nameA/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/nameA/.ssh/id_rsa.
Your public key has been saved in /home/nameA/.ssh/id_rsa.pub.
The key fingerprint is:
bb:3b:14:be:5d:45:ab:72:27:ec:93:21:c6:a3:7d:77 nameA@machineA
The key's randomart image is:
+--[ RSA 2048]----+
|                 |
|                 |
|                 |
|       .   .     |
|      .X. o .    |
|       .o.  +    |
|       .*+.o     |
|       +++C+o E  |
|      . +C++ .   |
+-----------------+
```

好了，准备工作就绪，我们开始建立信任关系：

```
[nameA@machineA]$ ssh-copy-id nameB@machineB
ssh: connect to host machineB port 22: Connection refused
```

新的错误提示又来了。原来我们的 B 机器的 sshd 的服务端口不是 22，而是 22000，但是 ssh-copy-id 命令却不知道这个信息，这可如何是好？

我们可以使用-p 选项来设置端口：

```
[nameA@machineA]$ ssh-copy-id nameB@machineB -p 22000
ssh: connect to host machineB port 22: Connection refused
```

还是不成，-p 选项完全没有被 ssh-copy-id 命令识别。

如果你用 man ssh-copy-id 可以看到，ssh-copy-id 根本就没有这个选项，该怎么办呢？

好吧，不卖关子了，其实解决办法一点也不复杂，只是用了一个小技巧，那就是：

```
[nameA@machineA]ssh-copy-id "-p 22000 nameB@machineB"
nameB@machineB's password:
[nameB@machineB]
```

大功告成，终于可以无密码登录了：

```
[nameA@machineA]$ ssh nameB@machineB -p 22000
[nameB@machineB]$
```

它其实是一个脚本文件

其实 ssh-copy-id 只是一个普普通通的脚本文件：

```
[nameA@machineA]$ which ssh-copy-id
/usr/bin/ssh-copy-id
[nameA@machineA]$ file /usr/bin/ssh-copy-id
/usr/bin/ssh-copy-id: POSIX Shell script text executable
```

如果你有兴趣，可以读一读这个脚本，只有短短 50 行，不过里面却有不少 Shell 编程技巧可以学习。

25 rsync 同步的艺术

同步初体验

如果你是一位运维工程师，那么你很可能会面对几十台、几百台甚至上千台服务器，除了批量操作外，环境同步、数据同步也是必不可少的技能。

说到“同步”，不得不提的是我们的同步利器 rsync，本文就来说说我从这个工具中体会到的同步的艺术。

我们经常这样使用 rsync：

```
[userA@machineA ~]$ rsync main.c machineB:/home/userB
```

1. 只要目的端（machineB）的文件内容和源端（machineA）不一样，就会触发数据同步，rsync 会确保目的端的文件保持和源端一致。
2. 但 rsync 不会同步文件的“modified time”，凡是有数据同步的文件，目的端文件的“modified time”总是会被修改为最新时刻的时间。
3. rsync 不太关注目的端文件的 rwx 权限。如果目的端没有此文件，那么权限会保持与源端一致。如果目的端有此文件，则权限不会随着源端变更。
4. 只要 rsync 有对源文件的读权限，且对目标路径有写权限，rsync 就能确保目的端文件同步到和源端一致。
5. rsync 只能以登录目的端的账号来创建文件，它没有能力保持目的端文件的输主和属组和源端一致。（除非你使用 root 权限，才有资格要求属主一致、属组一致）

-t 选项让修改时间也同步

我们经常这样使用-t 选项：

```
[userA@machineA ~]$ rsync -t main.c machineB:/home/userB
```

1. 使用-t 选项后，rsync 总会想着一件事，那就是将源文件的“modified time”同步到目标机器。
2. 带有-t 选项的 rsync，会变得更“聪明”，它会在同步前先对比两边文件的时间戳和文件大小。如果时间戳和文件大小都完全一致，那么就认定两边文件是一样的，于是，对这个文件就不会发起同步动作了。
3. 因为 rsync 的“聪明”，所以也会反被聪明误。如果目的端的文件的时间戳、大小和源端完全一致，但是内容恰巧不一致时，rsync 就发现不了了。这就是传说中的“坑”！
4. 对于 rsync 自作聪明的情况，解决办法就是使用-I 选项（字母 i 的大写形式）。

-I 选项踏实做人

我们经常这样使用-I 选项（字母 i 的大写形式）：

```
[userA@machineA ~]$ rsync -I main.c machineB:/home/userB
```

1. -I 选项会让 rsync 变得很乖巧，即它会挨个文件去发起数据同步。
2. -I 选项可以确保数据的一致性，代价便是速度上会变慢，因为我们放弃了“quick check”策略。
3. 无论情况如何，目的端文件的 modified time 总会被更新到当前时刻。

> 小科普，“quick check”策略，就是先查看文件的时间戳和文件大小，先排除一批认为相同的文件。

-v 选项让我们了解更多

这个选项，简单易懂，就是让 rsync 输出更多的信息。我们可以举一个例子：

```
[userA@machineA ~]$ rsync -vI main.c machineB:/home/userB
main.c

sent 81 bytes  received 42 bytes  246.00 bytes/sec
total size is 11  speedup is 0.09
```

增加越多的 v，就可以获得越多的日志信息。

```
[userA@machineA ~]$ rsync -vvvvt abc.c machineB:/home/userB
cmd= machine=machineB user= path=/home/userB
cmd[0]=ssh cmd[1]=machineB cmd[2]=rsync cmd[3]=--server cmd[4]=-vvvvte.
```

```
cmd[5]=. cmd[6]=/home/userB
opening connection using: ssh machineB rsync --server -vvvvte. .
/home/userB
note: iconv_open("ANSI_X3.4-1968", "ANSI_X3.4-1968") succeeded.
(Client) Protocol versions: remote=28, negotiated=28
(Server) Protocol versions: remote=30, negotiated=28
[sender] make_file(abc.c,*,2)
[sender] flist start=0, used=1, low=0, high=0
[sender] i=0  abc.c mode=0100664 len=11 flags=0
send_file_list done
file list sent
send_files starting
server_recv(2) starting pid=31885
recv_file_name(abc.c)
received 1 names
[receiver] i=0   abc.c mode=0100664 len=11
recv_file_list done
get_local_name count=1 /home/userB
recv_files(1) starting
generator starting pid=31885 count=1
delta transmission enabled
recv_generator(abc.c,0)
abc.c is uptodate
generate_files phase=1
send_files phase=1
recv_files phase=1
generate_files phase=2
send files finished
total: matches=0  hash_hits=0  false_alarms=0 data=0
generate_files finished
recv_files finished
client_run waiting on 14318

sent 36 bytes  received 16 bytes  104.00 bytes/sec
total size is 11  speedup is 0.21
_exit_cleanup(code=0, file=main.c, line=1031): entered
_exit_cleanup(code=0, file=main.c, line=1031): about to call exit(0)
```

-r 选项让文件夹递归同步

我们在第一次使用 rsync 时，往往会遇到这样的囧境：

```
[userA@machineA ~]$ rsync mydir machineB:/home/userB
skipping directory mydir
```

如果你不额外告诉 rsync 你需要它帮你同步文件夹的话，它是不会主动承担的，这也正是 rsync 的懒惰之处。

所以，如果你真的想同步文件夹，那么一定要加上-r 选项，即 recursive（递归的、循环的），像这样：

```
[userA@machineA ~]$ rsync -r mydir machineB:/home/userB
```

我们在上面的讲解中说过，如果时间戳和文件大小完全一致，只有文件内容不同，且你没有使用-I 选项的话，那么 rsync 是不会进行数据同步的。

现在提个问题：“因为在 Linux 的世界里，文件夹也是文件，如果这类文件（文件夹）也只有内容不同，而时间戳和文件大小都相同，那么 rsync 会发现么？”

这是个好问题。读者可以自己动手做实验，结论在这里告诉大家：

对于文件夹，rsync 是会明察秋毫的，只要你加了-r 选项，它就会恪尽职守地进入到文件夹里去检查，而不是只对文件夹本身做“quick check”的。

在这里，再教大家一个小技巧。因为文件夹传输时往往数据量会比较大，所以 rsync 提供了用于数据压缩的-z 选项。只要使用了这个选项，rsync 就会把发向对端的数据先进行压缩再传输。建议大家在网络环境较差的情况下使用。

软链文件如何处理

如果我们要同步一个软链接文件，猜猜 rsync 会提示什么？

```
[userA@machineA ~]$ ll
total 128
-rw-rw-r--  1 userA userA 11 Dec 26 07:00 abc.c
lrwxrwxrwx  1 userA userA  5 Dec 26 11:35 softlink -> abc.c

[userA@machineA ~]$ rsync softlink machineB:/home/userB
skipping non-regular file "softlink"
```

猜对了么，眼见为实，rsync 又无情地拒绝了我们。它一旦发现某个文件是软链接，就会无视它，除非我们增加-l 选项（字母 L 的小写）。

```
[userA@machineA ~]$ rsync -l softlink machineB:/home/userB
```

使用了-l 选项后，rsync 会完全保持软链接文件类型，原原本本地将软链接文件复制到目的端，而不会“follow link”到指向的实体文件。

如果就想让 rsync 采取 follow link 的方式，那就用-L 选项就可以了。读者可以自己试试效果。

权限保持也讲艺术

-P 选项的全名是“perserve permissions”，顾名思义，就是保持权限。

如果不使用此选项的话，rsync 是这样来处理权限问题的：

1．如果目的端没有此文件，那么在同步后会将目的端文件的权限保持与源端一致。

2．如果目的端已存在此文件，那么只会同步文件内容，权限保持不变。

如果你使用了-p 选项，则 rsync 会尽力让目的端保持与源端的权限保持一致，注意，仅仅是尽力而已。

而-g 选项和-o 选项，则是一对好兄弟，用来保持文件的属主（owner）和属组（group），作用清晰明了。不过需要注意的一点是，改变属主和属组的权限，往往只有管理员才有。

-a 选项最霸道

1．-a 选项是 rsync 里比较霸道的一个选项，因为一旦使用了-a 选项，就相当于使用了-rlptgoD 这一堆选项。以一敌七，唯-a 选项也。相信在看了前文之后，你应该可以很轻松地理解这七个选项的作用了。

2．-a 选项的学名叫作 archive option，中文叫作归档选项。使用-a 选项，就表明你希望采取递归方式来同步，且尽可能地保持各个方面的一致性。

3．但是-a 选项也有“阿克琉斯之踵”，即-a 无法同步“硬链接”的情况。如果有这方面需求，需加上-H 选项。

小科普：“阿克琉斯之踵”来自于一个希腊神话故事，一般是指致命的弱点或要害。

--delete 选项三姐妹

我们要介绍的这三个选项都是和“删除”有关的：

1．--delete：如果源端没有此文件，那么目的端也别想拥有，删除之。（如果你使用这个选项，则必须搭配-r 选项）

2．--delete-excluded：专门指定一些要在目的端删除的文件。

3．--delete-after：默认情况下，rsync 是先清理目的端的文件再开始数据同步。如果使用此选项，则 rsync 会先进行数据同步，都完成后再删除那些需要清理的文件。

看到这么多的 delete，你是否有点肝颤？的确，在 rsync 的官方说明里也有这么一句话：

```
This option can be dangerous if used incorrectly!
It is a very good idea to run first using the dry  run  option
(-n) to see what files would be deleted to make sure
important files aren't listed.
```

从这句话里，我们学到了一个小技巧，那就是-n 选项。这是一个吓唬人的选项，它会用受影响的文件列表来警告你，但不会真的去删除，这就让我们有了确认的机会和回旋的余地。一起来看看实际用法吧：

```
[userA@machineA ~]$ rsync -n --delete -r . machineB:/home/userB/
deleting superman/xxx
deleting main.c
deleting acclink
```

--exclude 选项和--exclude-from 选项

如果你不希望同步一些东西到目的端的话，则可以使用--exclude 选项来隐藏。rsync 还是很重视大家隐私的，你可以多次使--exclude 选项来设置很多的“隐私”。

如果你要隐藏的隐私太多的话，那么在命令行选项中设置会比较麻烦。rsync 很体贴，它提供了--exclude-from 选项，让你可以把隐私一一列在一个文件里，然后让 rsync 直接读取这个文件就好了。

--partial 选项

这就是传说中的断点续传功能。默认情况下，rsync 会删除那些传输中断的文件，然后重新传输。但在一些特殊情况下，我们不希望重传，而是续传。

在使用中，我们经常看到有人使用-P 选项，这个选项其实是为了偷懒而设计的。以前人们总是要手动写--partial --progress，觉得太费劲了，倒不如用一个新的选项来代替，于是-P 应运而生了。有些读者可能会问，--partial 我知道作用了，可--progress 是做什么的呢？为什么很多人要使用它呢，它有那么大的吸引力吗？

--progress 选项

使用这个选项，rsync 会显示传输进度信息。有什么用呢？rsync 给了一个很有意思的解释：

```
This gives a bored user something to watch.
```

好了，写了这么多，纸上得来终觉浅，绝知此事要躬行，到底从哪个选项实践起呢？可以先试用一下--progress 解解闷，或许是个不错的选择，哈哈。

26 其实你不懂wget的心之———下载文件

wget 是何方神圣

wget 用英语来描述就是 the non-interactive network downloader，中文称之为非交互的网络下载器。

wget 支持 HTTP、HTTPS 和 FTP 协议的下载，其中也包括通过 HTTP 代理的下载。看起来好像支持的协议并不多，但其实这已经足够了，通常我们是很少使用除这些协议之外的其他协议来进行下载的。

介绍完 wget 的基本功能，下面来为大家介绍下 wget 的特性和优势：

- wget 能够跟踪 HTML 文件和 XHTML 文件，从而使得你可以下载整个站点的文件，然后离线阅读。当然这个功能用起来要小心一些，因为几乎所有站点都会包含外链，一旦你用 wget 进行整站的下载，它也会傻傻地去下载外链站点的内容，这样一来，子子孙孙无穷尽也。
- wget 是个非常遵守 Robot Exclusion Standard（robots.txt）标准的工具。
- wget 支持慢速网络下载和不稳定网络的下载。当因为网络不稳定而导致下载失败时，wget 会重试直到把目标文件下载成功为止。
- wget 支持断点续传功能，当你下载的很大的文件在中途失败时，这项功能会很有用处。

一起来下载一个文件吧

说了这么多 wget 的特性和功能，不如来亲自体验一下。我们还是按照惯例，把 wget 的最简单用法先介绍给大家。

首先试着用 wget 工具来下载“运维帮”的 logo 图片：

```
[roc@roclinux ~]$ wget http://www.yunweibang.com/yunweibang.jpg
--2016-04-18 22:52:23--  http://www.yunweibang.com/yunweibang.jpg
正在解析主机 www.yunweibang.com... 121.42.192.215
```

```
正在连接 www.yunweibang.com|121.42.192.215|:80... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度：8413 (8.2K) [image/jpeg]
正在保存至：“yunweibang.jpg”

100%[=====================================================================
========================>] 8,413      --.-K/s   in 0s

2016-04-18 22:52:24 (238 MB/s) - 已保存 “yunweibang.jpg” [8413/8413])
```

下载成功！我们来欣赏下“运维帮”那经典的小狗形象，如图 8 所示。

图 8　运维帮 Logo

说句题外话，写这本命令文集，还是因为参加了“运维帮”举办的一场技术讲座，偶获灵感，才决定写这样一套书的。感谢这只“给了我灵感”的可爱小狗。

说说 wget 的一些潜规则

提到潜规则，我们先来说一下有关配置文件的潜规则吧。当你使用 wget 时，请务必查看/etc/wgetrc 文件和家目录下的.wgetrc 文件，先搞清楚里面都设置了什么，再来使用 wget 命令不迟。因为这两个文件会配置 wget 的一些默认行为，这对大家使用 wget 可能会有比较大的影响。

举一个实际的例子，你就会有所体会啦。

wget 命令有一些选项可以接受用逗号隔开的参数，比如-X（大写的 x）选项，它用来设定“不希望下载”的目录列表。你可以用逗号把不希望下载的目录一个一个列出来，比如“wget -X wukong,bajie”，这样 wget 心里就有数了，它知道 wukong 目录和 bajie 目录都是不用下载的。

如果这些目录是你长期不希望下载的，那么你完全可以在.wgetrc 文件中设置“长期不希望下载”的目录列表，格式是这样的：

```
exclude_directories=wukong,bajie
```

于是，你不必在 wget 命令上设置，就可以实现“不下载”wukong 和 bajie 目录。

当然，如果某天你下载文件时，发现总有几个目录下载不下来，那么你应该想到，有可能是其他人设置了.wgetrc 造成的。当然你还要再去/etc/wgetrc 文件中确认一下你的假设。

还有一个小技巧，也可以避免这种误会（别人设置了 exclude_directories，而你却不知道），那就是在你使用 wget 时，这样来写：

```
wget -r -X '' -X wukong,bajie ftp://localhost
```

使用-X"的目的就是去除.wgetrc 和/etc/wgetrc 的作用，然后再用-X wukong，bajie 设置，就可以踏踏实实地保证不下载 wukong 和 bajie 目录，而其他目录绝不会受影响。

有些同学可能会问两个配置文件之间的覆盖关系，实际上，wgetrc 和-X 和/etc/wgetrc 的设置是属于平等关系，三者在使用时会进行并集。而在设置了-X '' 时，就完全去掉了.wgetrc 和/etc/wgetrc 对目录的限制作用。

好了，通过这个例子，大家不仅学习到了配置文件的作用、-X 选项的作用，而且还知道了由此带来的一个潜规则，是不是感觉受益颇多！

wget 目录下载和后台执行

如果你认为 wget 只能下载单个文件，那你可是太小看 wget 了，使用-r 选项我们就可以轻松实现多级目录的“递归下载”啦。

而有时候，如果下载的目录和文件太多，我们就希望 wget 到后台去下载。可以使用--background 选项，这时启动的 wget 会立即进入后台执行。

如果没有使用-o 选项设置“下载日志文件”位置的话，默认是将其记录在当前目录的 wget-log 文件中，其实这个日志就是当初输出到屏幕上的那些内容而已。需要注意的是，即使你的远程终端连接被 Ctrl+D 或 exit 了，也不会影响到 wget 的后台执行。

我想避开 robots 协议

虽然这不是一个好主意，但是不可否认的是，wget 的确可以做到。

在下载网站内容时，如果遇到 robots.txt 封禁的话，我们可以借助--execute robots=off 来避开 robots.txt 的封禁。

好了，本文让大家认识了 wget 命令，也了解了 wget 命令的一些小技巧和潜规则，接下来的一篇，我们会针对 robots 协议做专门的场景再现。

27 其实你不懂wget的心之二——躲避封禁

上一篇文章中我们提到了 wget 可以避免 robots.txt 封禁的事情。这篇文章我们就来做个试验，让大家亲自体验一下。

第一步：搭建一个临时的 Web 服务器，将端口号设置成 61212。网页文件所在路径为/home/roc/program/apache/htdocs，我们简称为 htdocs 目录。

第二步：在 htdocs 目录中，建立一个 index.html 文件，内容如下：

```
$ cat -n index.html

1  <html>
2         <head>
3                <title>roclinux</title>
4         </head>
5         <body>
6                <ul>
7                       <li><a href=roclinux-1.html>roclinux-1</li>
8                       <li><a href=roclinux-2.html>roclinux-2</li>
9                </ul>
10         </body>
11  </html>
```

第三步：手动建立如下文件，当然也包括 robots.txt 文件：

```
$ ls -1
index.html
robots.txt
roc.html
roclinux-1-1.html
roclinux-1-2.html
roclinux-1.html
roclinux-2-1.html
roclinux-2-2.html
roclinux-2.html
```

根据文件编号，你应该能看出这些文件之间的从属关系。

第四步：建立简单的 robots.txt 文件：

```
$ cat robots.txt
```

```
User-agent: *
Disallow: roclinux-2.html
```

这个文件的内容就是要屏蔽 roclinux-2 文件的下载和 spider 的来访。

第五步：用 wget 来下载这个测试站点：

```
$ wget -r http://my-test.cn:61212/

$ ls -1
index.html
robots.txt
roclinux-1-1.html
roclinux-1-2.html
roclinux-1.html
```

看！下载到的文件中包含了 robots.txt 文件，但没有包含 roclinux-2 及其从属的文件。

可见，robots.txt 生效了，wget 遵守了 robots.txt 的规则。

第六步：我们一起来突破 robots.txt 限制。

使用--execute 选项就可以啦：

```
$ wget -r --execute robots=off http://my-test.cn:61212/

$ ls -1
index.html
roclinux-1-1.html
roclinux-1-2.html
roclinux-1.html
roclinux-2-1.html
roclinux-2-2.html
roclinux-2.html
```

看，roclinux-2 系列的文件也都顺利地下载下来了。

28 其实你不懂wget的心之三——下载目录

我想下载整个文件夹

当深入学习 wget 时，你会发现它的选项实在是太多了，错综复杂乱如麻。本文中，我们力争做到专注，只针对常用的目录选项展开讨论。

-r 选项就是用于下载远程的文件夹的，但是情况并没有想象得那么简单，对于 ftp 服务来讲，假如你使用了下面的命令来下载文件夹的话：

```
[roc@roclinux ~]$ wget -r ftp://my.test.server:/home/roc/img
```

那么，实际在当前目录下会生成 my.test.server/home/roc/img 这样的多层级目录结构，可见直接使用-r 选项，默认会创建“域名和绝对路径”组成的目录结构。虽然我们将数据成功下载到本地了，但这或许并不是我们的初衷，至于更优雅的解决办法，我们继续往下看。

-nH 选项来帮忙

-nH 选项，即--no-host-directories。通过上例我们已经知道了在使用 wget -r 命令下载目录时，默认会创建以域名 my.test.server 命名的文件夹。而使用-nH 选项后就可以避免这种默认行为。

所以，当你用如下命令下载文件时，只会在当前目录下创建 home/roc/img 目录结构，而原来的 my.test.server 文件夹已经不见了。

```
$ wget -r -nH ftp://my.test.server:/home/roc/img
```

域名去掉了，但长长的目录路径仍然还在，如果我只想下载 img 单个文件夹，不希望掺杂着前面的路径，该怎么做呢？

用--cut-dirs=number 选项继续优化

这个选项很常用，它表示下载数据时，在本地创建目录时，忽略前面多少层的目录结构。

我们拿 ftp://my.test.server:/home/roc/img 为例，如果只使用-r 选项，那么本地会创建 my.test.server:/home/roc/img 目录结构。如果再加上-nH 选项，则留下来的目录结构是 home/roc/img。此时就可以用--cut-dirs 来继续清理前面的目录了，如表 6 所示。

表 6　nH 选项和--cut-dirs 配合效果

情　　况	结　　果
只用-r 选项	my.test.server:/home/roc/img
-nH	home/roc/img/
-nH --cut-dirs=1	roc/img/
-nH --cut-dirs=2	img/
-nH --cut-dirs=3	.
-cut-dirs=1	my.test.server/roc/img/

从表中大家应该知道-nH 和--cut-dirs 两个选项互相配合的效果了吧。

如何实现平铺效果

-nd 选项，即--no-directories，从字面来看，似乎是“不允许有目录”的意思。

你没猜错，当我们下载远程的数据时，可以要求 wget 只下载文件，不下载文件夹，所有下载的文件都平铺在当前目录下。

这时，善于思考的同学会提出一个问题：“如果下载到不同路径的同名文件的话，那么用-nd 选项岂不是会造成同名文件覆盖的问题？”。答案是不会的，因为 wget 在下载文件时，如果当前目录下有同名文件，则会默认在新下载的文件后加上“.1”“.2”等序号标识，以示区别。

让世界充满文件夹

-x 选项，即--force-directories，这个选项和--no-directories 的效果是完全相反的。

--no-directories 选项是要求绝对不能下载和创建任何文件夹，同时所有文件都平铺在当前目录中。

而 --force-directories 选项则要求处处都要有文件夹，即使是 wget -x http://fly.srk.fer.hr/home/robots.txt 这样下载单独普通文件的命令，也会在当前目录下创建 fly.srk.fer.hr/home 目录结构，然后将 robots.txt 文件下载到 fly.srk.fer.hr/home 里面。

以协议之名

--protocol-directories 选项的作用是先创建一个以协议名为名称的文件夹，例如：

```
$ wget -r --protocol-directories ftp://my.test.server:/home/roc/img
```

则会创建目录结构 ftp/my.test.server/home/roc/img。

这个选项，对于那些希望通过协议类型来区分数据的场景比较有用。

29 其实你不懂 wget 的心之四——体贴的选项

我想实现自动重试

这个段落要讲的是-t 选项，即--tries=number，用于设置 wget 下载时失败重试的次数，当设置为 0（数字零）或 inf 时表示无限次重试。默认的重试次数是 20 次。

不过 wget 也不是在什么情况下都会傻傻地重试，例如，在发生“connection refused”或“not found”时，wget 会立即退出，不会再继续进行重试。只有在网络出现不稳定或者大数据下载出现异常时，才会触发重试机制。

小 o 和大 O

标题很迷惑，虽然名字相似，但可以提前告诉大家，其实小 o 和大 O 是解决完全不同的问题的，在功能上并没有什么交集。

我们先来说说-o 的作用。-o 选项，即--output-file=logfile，wget 运行过程中输出到标准输出的内容都会被写到所设置的 logfile 文件中。如果你希望把 wget 下载过程中的这些的信息保留下来，那么-o 选项正好适合你。

-O 选项，即--output-document-file，表示 wget 下载的所有文件的内容会被依次追加写到所设置的 file 文件中，而不会创建原本该下载的文件。

在下载单独文件时使用-O 选项，则可以实现另一个效果，那就是直接强制写入所设置的文件中，避免 wget 下载本地同名文件时默认将下载的文件写到以“.1”为后缀的文件中。

-nc 选项不好理解

-nc 选项，即--no-clobber 选项，这个选项的作用并不好描述，我们需要花些篇幅来

为大家仔细介绍，如果你一遍没有读懂，请耐心再读两三遍。

在讲解-nc 选项前，我们先为大家介绍一下-N 选项，即--timestamping，它表示开启时间戳机制，wget 会先比较远程和本地文件的时间戳，然后只将远程时间戳更新的文件下载到本地。

之所以要先介绍一下-N 选项，是因为下面介绍-nc 选项时，会涉及-N 选项的知识。好了，我们言归正传，来介绍-nc 选项啦。

在同一个目录中，如果一个文件被多次下载，那么 wget 的处理方式会取决于几个重要选项，这其中就包括了-nc 选项。

当多次下载同一个文件时，本地同名文件或者被覆盖，或者被重写，或者被保护，这都是有可能的。

当使用 wget 多次下载同一个文件，且不使用-N、-nc 或-r 时，那么 wget 会默认在第二次下载时自动在文件名后加上“.1”后缀，第三次下载时加上“.2”后缀，以此类推。

但当我们使用-nc 选项时，wget 不会使用“后缀.1、后缀.2”这样的策略，而是拒绝下载同一文件（即使文件内容是更新的了）。这个功能很实用，我们可以使用-nc 避免相同文件被多次下载。

当 wget 命令使用了-r 选项，但没使用-N 选项或-nc 选项时，假如重新下载同名文件，若远程文件的修改时间比本地的新，那么 wget 会选择覆盖当前目录已有的这个旧文件。假如此时我们加上了-nc 选项，则可以禁止 wget 这样去做。（但当远程文件的修改时间并不新时，wget 会拒绝下载。）

当 wget 使用-N 选项时，是否下载同名文件，完全取决于远程文件和本地文件的时间戳以及文件大小。-nc 选项是不允许和-N 选项同时设置的。如果你同时使用了-N 和-nc 选项，就会得到这样的错误提示“Can’t timestamp and not clobber old files at the same time.”。

-c 选项实现断点续传

-c 选项，即--continue 选项，就是大名鼎鼎的“断点续传”。无论你之前使用哪个下载工具下载了一半的文件，都可以用 wget 来继续下载此文件。比如：

```
[roc@roclinux ~]$ wget -c ftp://sunsite.doc.ic.ac.uk/ls-lR.Z
```

当前目录已有一个 ls-lR.Z 文件存在，wget 将假定这是一个下载了一半的文件，然后提取本地文件的文件大小，并根据此值请求从远程文件的相应文件大小处开始继续下载。

你会发现，其实 wget 的此断点续传策略是有隐患的，因为如果远程文件的开头部分被进行了修改，则 wget 在进行断点续传时是意识不到这一点的，它只会傻傻地从已传文件大小之后的部分继续下载。所以使用-c 选项断点续传之后，务必进行 md5 校验。

限速天天见

限速是运维场景中很关键的技能，因为限速可以有效控制源端和目的端服务器的 I/O 和 CPU 负载情况，避免因为突发的大流量导致资源耗尽。

--limit-rate=N 选项便是用于下载下载限速的，将速度限制在 N bytesBytes/second，当然也可以用 k 或 m 单位 k/m 来作为单位表示。例如--limit-rate=20k 将会限制速度在 20KB/s。

注意，wget 实现限速的原理是在一次网络读取动作之后 sleep 一个特定时间段，以让平均的网络读速度降到限制值，这个策略最终会使 TCP 传输速度降到限制值左右。所以在传输超小文件时，可能无法达到限速的作用。

-w 选项和--waitretry=seconds 选项

-w 选项，即--wait=seconds 选项，用于设置 wget 每两个请求之间间隔的秒数。这个选项很有用处，可以降低远程服务器的负载。除了可以直接设置秒数外，还可以加上 m 表示分钟、h 表示小时、d 表示天。

而--waitretry=seconds 选项则是用于设置请求重试秒数的。wget 采用的是线性递增等待的方式，如果你设置的是 10 秒，那么第一次请求失败后，会等待 1 秒；第二次请求失败后会等待 2 秒；直到最后达到 10 秒等待时间为止。所以当到达最后一次时，时间已经过了 1+2+…+10=55 秒。

好了，wget 系列的内容全部结束了，我们从入门到封禁、从目录到选项，把 wget 最常用的和最有趣的知识点都介绍了一遍。wget 在运维场景下十分常用，相信会对你的学习和工作有所帮助的。

进程和性能篇

在进程和性能篇中，我们将为大家带来 22 篇文章，所有内容都是围绕着进程查看和服务器性能展开的，包括：

在这里，你不仅可以解到 free 原来有这么多学问，了解 top 的详细用法，还可以跟随 lsof 一起去侦破三桩悬案！

让我们现在就开始进程和性能篇的学习之旅吧！

1 uptime 给机器记考勤

老运和小维

人物介绍

老运：资深运维工程师，从事运维工作七年。

小维：名牌大学毕业生，现刚加入运维部。

故事开始

小维：老运，现在我管理几千台机器，每台机器的硬件信息我现在都了解得差不多了，但它们的运行状态，我是一点都不了解？

老运：那你想了解哪些方面的信息呢？

小维：我就是想每天一早巡检时，能快速了解每台机器的负载情况，还有机器的开机时长，如果开机超过 100 天，我们公司规定是要安排手动重启一次的。

老运：使用 uptime 命令就可以搞定。

uptime 命令

uptime 命令，有两大功能：一个是查看机器的开机时长，另一个是查看 CPU 负载情况。

下面来看看 uptime 命令的实际运行效果：

```
[root@roclinux ~]# uptime
 22:36:58 up 11:12,  1 user,  load average: 0.00, 0.00, 0.00
```

其含义如表 7 所示。

表 7　uptime 命令输出

输出项	解　　释
22:36:58	系统当前时间
up 11:12	主机已运行时间，时间越长，说明机器越稳定
1 user	用户连接数，是总连接数而不是用户数
load average: 0.00, 0.00, 0.00	系统平均负载，统计最近 1、5、15 分钟的系统平均负载

小维：老运，我有一个不明白的地方，到底什么叫系统平均负载呢？

老运：系统平均负载，就是在特定时间间隔内运行队列中的平均进程数。

小维：我晕，还是不明白呀……

老运：好吧，我们来专门说说这个知识点。

什么是系统平均负载

系统平均负载，是指在特定时间间隔内运行队列中的平均进程数。

首先讨论单核 CPU 情况下的系统平均负载。我们尝试用“坐大巴”的场景，来说明这个事情。

- 一个单核 CPU 可以形象地比喻成一辆大巴车。
- 一般来说，大巴车的载客数是一定的，比如 30 人。当大巴车载客营业时，如果载客 20 人的话，这时乘客会感觉车内很宽松；当载客 30 人的话，这时刚刚好，每一个人都有座位坐，但已经让人感觉拥挤了。当继续有乘客上车的话，就会有人站在车的过道上，让人感觉车内拥挤不堪。
- 那如何体现车的拥挤程度呢？ 这就要用到平均负载这个量化指标了。
  - ✧ 0.00 表示车内一个人都没有，这时车是空载。
  - ✧ 1.00 表示车内乘客每个人都有座位坐，这时车是满载。

超过 1.00 表示车内乘客有人没有座位坐，这时司机要注意了，如果春运期间，被警察叔叔发现的话是要罚款的。

上面的平均负载可以很好地体现车内的拥挤情况，而在计算机中，平均负载也能很好地体现系统的繁忙程度。但和大巴不同的是，车内平均负载为 1 时仍然是安全的，而计算机世界里，按一般的经验来看，单核负载在 0.7 以下是安全的，超过 0.7 就需要进行优化了。

如果大家理解了单核 CPU 的系统平均负载，那么理解多核 CPU 的系统平均负载就

会容易得多。多核 CPU，就好像我们有多辆大巴车，承载能力和运力都成比例增加。我们设服务器具有 *N* 个 CPU 核，那么负载值如果小于 *N*，则认为服务器仍然在可承受范围内，否则，就是过载状态啦。

小维：原来如此，明白了，以后就用这招了。

老运：哈哈，最后再附送你一个锦囊。

```
[root@roclinux ~]# cat /proc/loadavg
0.00 0.00 0.00 1/166 2645
```

原来，我们还可以从/proc/loadavg 文件中来获取系统平均负载的信息。

除了前 3 个数字表示 1、5、15 分钟内的系统平均负载情况外，后面的分数（1/166）中，分母（166）表示系统的总进程数，分子（1）表示正在运行的进程数；而最后一个数字（2645）则表示最近一个启动运行的进程的 ID。

好了，在本文中，我们通过老运和小维的一系列对话，为大家介绍了 uptime 命令的相关知识，也了解了查看系统负载的一个方法。系统负载是一个很大的话题，还有很多命令可以从各种维度查看服务器的负载情况，如果你还不清楚，那请继续往下阅读，惊喜还在后面呢！

2 内存不决问 free

为什么叫作 free 呢

刚刚接触 Linux 时，看到 free 命令，完全想不到它会和内存扯上关系，毕竟 free 在英语里的含义是“免费的、空闲的、自由的”。

为此，我还特意发邮件询问了 free 命令的维护者 albert，但是很遗憾，他也不知道当时选择这个单词的典故和轶事。

我猜，因为 Linux 发明之初，内存是极其稀缺的资源，内存都是按照 KB 来计价的，当时的工程师们非常不爽，所以用 free 命名内存，希望有朝一日，内存资源可以变成“免费、自由之物”，听着很有道理吧，哈哈！

free 的最常用用法

最最常用的用法，当然就是直接输入 free 啦，输出内容则以 KB 为单位：

```
[roc@roclinux ~]$ free
              total        used        free      shared  buffers     cached
Mem:       3104152     2088116     1016036           0     301520     505984
-/+ buffers/cache:     1280612     1823540
Swap:            0           0           0
```

如果想让数字更可读，大家往往会加上-m 选项，让输出内容以 MB 为单位展示：

```
[roc@roclinux ~]$ free -m
              total        used        free      shared  buffers     cached
Mem:          3031        2039         992           0         294         494
-/+ buffers/cache:        1250        1780
Swap:            0           0           0
```

从 free 的输出内容中，可以看出，这台服务器拥有 3GB 内存容量，对于一台入门级服务器来说，已经不算小了。

要慎用-g 选项

有些同学比较喜欢使用-g 选项，让输出内容以 GB 为单位显示。这样虽然可以增加可读性，但却存在不小的隐患：

1. 首先，-g 选项并非是官方支持的选项，你会发现在 man free 时是看不到这个选项的，所以不确认在所有的 free 版本中都支持。
2. 最重要的是，-g 选项会采用向下取整的方式显示内存容量，就如本文给出的例子，原本 3031MB 内存容量，换算后变为 2.96GB，最后会显示成 2GB，这会在很大程度上误导用户。

就像下面这样，本来服务器的内存总量是 3GB，但是这里却显示成了 2GB，是不是很坑人呢？

```
[roc@roclinux ~]$ free -g
             total       used       free     shared    buffers     cached
Mem:             2          2          0          0          0          0
-/+ buffers/cache:          1          1
Swap:            0          0          0
```

想查看最精确的内存容量

无论用-k 选项、-m 选项，还是-g 选项，free 命令的输出策略都是向下取整，所以如果你想查看一台服务器最精确的内存容量，那么请使用-b 选项，这会让 free 命令以最小的 byte 为单位显示内存使用状况：

```
[roc@roclinux ~]$ free -b
             total       used       free     shared    buffers     cached
Mem:    3178651648 1892573184 1286078464          0  293597184  428122112
-/+ buffers/cache: 1170853888 2007797760
Swap:            0          0          0
```

大家应该看到 free 命令输出的 shared 这个指标了吧，它的值总是 0，这是怎么回事呢？我们继续往下看。

说说 shared 被弃用这事儿

在执行 man free 时，会有对 free 命令的简单介绍：

```
free displays the total amount of free and used physical and swap memory
in the system, as well as the buffers used by the kernel.  The shared memory
column should be ignored; it is obsolete.
```

里面有一个 obsolete，我专门查了百度翻译，意思是：

```
obsolete
adj.
废弃的；老式的，已过时的；[生]已废退的；
n.
废词；被废弃的事物；
vt.
淘汰；废弃；
```

显而易见，这一段的最后一句话，就是在说有关 shared 的事情，那就是 shared 已经被正式弃用，大家不必花时间研究这个名词啦。

在此之前，shared 是指系统中多个进程所共享的内存容量，它可以用来反映系统中可被复用的内存量。

但这个指标对于反映整个系统的内存使用情况并没有太多指导意义，所以在新版的 free 中，shared 被弃用，并总是显示为 0。

buffers、cached 傻傻分不清楚

在我参与的所有技术面试中，即便是工作多年的社招同学，也很少有人能说清楚 buffers 和 cached 的含义和区别。

接下来，我们就重点说说 buffers 和 cached 的含义和区别。

首先，要和大家明确的一点是，无论 buffers 还是 cached，都是属于内存的一部分，了解这一点格外重要，它们和空闲内存、已用内存一起构成了整个内存容量。

众所周知，硬盘和内存的读写速度有着本质的区别，目前消费级市场最快的 DDR4 内存，读写速度大概在 60GB/s 量级，而目前最快的 SSD 固态硬盘的读写速度也仅仅是 600MB/s 的水平。可见，内存和硬盘之间，存在着 100 倍左右的速度差。

当有大量数据要从内存写入硬盘时，为了防止读写数据速度的巨大差距导致的时间等待，伟大的科学家们想到了一个策略，那就是在内存中创造一个叫作 buffers 的内存区域，数据不再直接“缓慢”地写入硬盘，而是先写入 buffers 中，然后再在后台慢慢地写入硬盘中，这样对于应用程序来说，就不需要再等待数据完全写入硬盘，就可以去做其他事情了。这个策略一经发明，大大提升了应用程序的工作效率。

那么，cache 又是什么呢？我们的故事继续。

大家已经知道，内存和硬盘之间的速度差距巨大，所以，系统设计的一个重要原则就是“要尽量减少内存从磁盘读数据的次数”。毕竟在内存看来，每次从硬盘读数据，还是挺慢的，所以，科学家们又发明了一个叫作 cache 的内存区域。从硬盘读取的内容，往往会暂存在 cache 里，下次如果又读到了，那么就不用再找硬盘要了，而是直接从 cache 里拿就好了。从经验来看，一个数据被重复读取的概率，还是挺大的，所以 cache 对于加速数据读取，起到了非常关键的作用。

细心的同学可能会说，“你讲的是 cache，可 cached 又是什么呢”。是的，cache 是一个名词，表示那块专用的内存区域中的内容，而 cached 则表示“被缓存住的”，是一种状态。相信学过英语的同学，应该可以理解两者的异同。

好了，通过故事性的描述，大家应该已经了解了 buffers 和 cached 的含义和区别了。

更深层次理解 buffers 和 cached

为了让求知欲更强的小伙伴可以对 buffers 和 cached 理解得更深入，我们再补充一些高级解释。

free 命令中的 buffers 和 cached 值，是读取自/proc/meminfo 文件中的对应值。而/proc 中的绝大部分内容是 Linux 内核来控制和更新的。

从 Linux 内核源码中，可以看到 buffers 的值是由 nr_blockdev_pages()函数来产生的，cached 的值是由 global_page_state(NR_FILE_PAGES) – total_swapcache_pages – i.bufferram 计算得来的。感兴趣的同学，可以深入到 fs/proc/meminfo.c 中继续研究。

从源码分析可以得出更专业的结论，即：

- buffers 是块设备 I/O 相关的缓存页。
- cached 是普通文件相关的缓存页。

好了，有关 buffers 和 cached 的知识就说到这里，相信应该足够 Linux 工程师用了。

free 命令输出里的秘密

以往每次面试，我都会问面试者有关 free 命令的问题，但是几乎没有人能准确说出 free 输出内容中各种名词的含义，以及它们对服务器的影响。

所以，接下来，我们就来攻克这道难关，让大家以后再遇到类似问题时，可以信心满满，对答如流。

```
[roc@roclinux ~]$ free -m
             total       used       free     shared    buffers     cached
Mem:          3031       2248        783          0        280        410
-/+ buffers/cache:       1557       1474
Swap:            0          0          0
```

这是一个最典型的 free 命令输出的内容。为了让大家更直观地学习和理解，我们通过图 9 来展示个中关系。

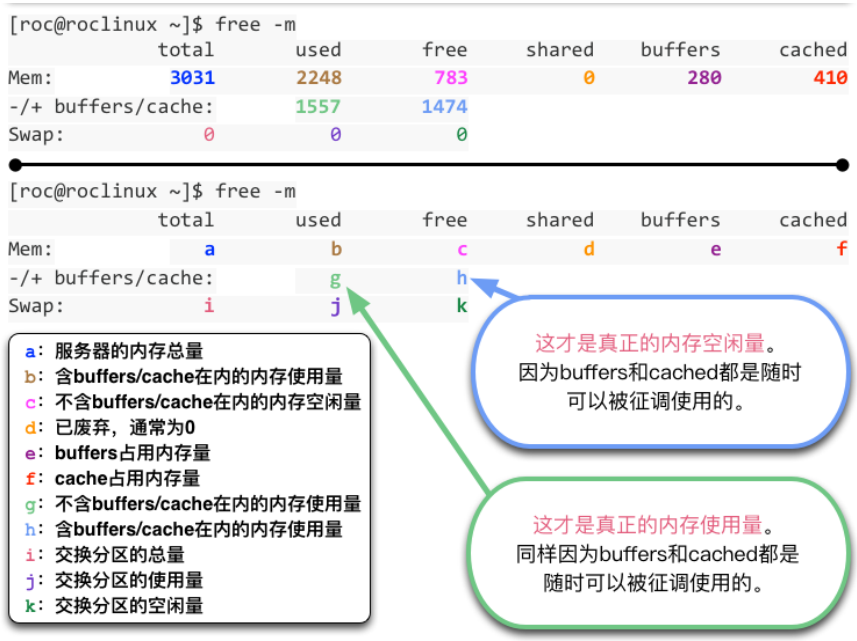


图 9　free 命令输出的内容间的关系

-o 选项的小故事

我们先直观地看一下-o 选项的作用：

```
[roc@roclinux splan]$ free -om
             total       used       free     shared    buffers     cached
Mem:          3031       1705       1326          0        282        407
Swap:            0          0          0
```

发现有什么不一样了么，如果没有的话，那么我们玩一次“大家来找茬”，来看一下没有-o 选项时的差异：

```
[roc@roclinux splan]$ free -m
             total       used       free     shared    buffers     cached
Mem:          3031       1715       1315          0        282        408
```

```
-/+ buffers/cache:       1025       2005
Swap:            0          0          0
```

揭晓答案，当使用-o 选项时，命令输出中“-/+ buffers/cache:”这一行就消失了。

原来，早期的 free 命令中的 used 和 free，就是第一行的样子，即 1715 和 1315。读了上文的同学一定知道，1315 里是没有包括 buffers 和 cached 的。这就造成了一个问题，内存空闲量有时候会看起来非常少，因为有可能大部分内存都被 buffers 和 cached 占据了。这时工程师会误认为空闲内存已经很少了，从而造成不必要的恐慌。

为了降低 free 命令造成误解的频率，提高其可读性和易用性，在后来的版本中，就默认加入了“-/+ buffers/cache”一行，其中的 free 值（2005）便是增加了 buffers 和 cached 的，而对应的 used 值（1025）则是减去了 buffers 和 cached 的。这就能更真实地反映“内存使用量”和“内存空闲量”了。

而为了做到与老版本的命令相兼容，所以允许用户使用-o 选项来不显示“-/+ buffers/cache”。

好了，关于 free 命令的各种故事，我们就讲到这里了。相比于网络上的各种 free 文章，本文应该算是从基础到高级，讲解最为全面的文章之一了。我们为大家制作的 free 命令的图解，大家可以贴到自己的工位，便于查看。

3 用好 SWAP 的空间

摩尔定律的预言

还记得摩尔定律么，摩尔定律是 Intel 创始人戈登·摩尔提出来的，意思是说：

> 当价格不变时，集成电路上可容纳的元器件的数目，约每隔18～24个月便会增加一倍，性能也将提升一倍。

换言之，每一美元所能买到的电脑性能，将每隔 18～24 个月翻一倍以上。

遥想 1999 年，一条 64MB 内存的价格曾达到过 1300 元人民币，而 2016 年，一条 8GB 的高性能内存的价格也就是 359 元人民币。

如此明显的性能提升、价格下降，真的是印证了摩尔定律的预言。

关于单位存储成本这事儿

现如今，虽然存储成本已经非常低了，但是与硬盘存储成本相比，内存存储成本仍然是相对较高的，我们来看表 8。

表 8　存储成本

类　别	品　牌	容　量	价　格	单位存储成本
SATA 机械硬盘	西部数码	1TB	339 元	0.3 元/GB
SSD 固态硬盘	饥饿鲨	240GB	399 元	1.7 元/GB
DDR3 内存	金士顿	8GB	359 元	45 元/GB

从表 8 中可以看出，内存成本是机械硬盘成本的 150 倍，是固态硬盘成本的 26 倍。

如此明显的成本差距，会让我们不由得思考一个问题：如何能在“高成本资源”短缺的情况下，把“低成本资源”的利用率提升上来呢？

SWAP 应运而生

正是由于内存资源十分昂贵，在内存不足时，工程师们才想到了更大限度利用硬盘资源的诸多方案，其中应用最广、方案最成熟的便是 SWAP 技术了。

SWAP，中文名称是“交换分区”。

如果你对 Windows 还算熟悉的话，那么你应该听说过“虚拟内存”吧。Windows 中的虚拟内存和 Linux 中的交换分区，是类似的思路和解决方案，都是当内存使用量不足时，系统将一部分硬盘空间虚拟成内存，来缓解内存使用紧张的问题。

交换分区的工作原理简述如下：

- 当 Linux 系统发现物理内存使用量不足时（那到底如何定义“不足”呢，留个悬念，后面揭晓），就会选择内存中较长时间没有被访问和更新的内存数据，将这些内存数据临时写到 SWAP 中，并释放内存中相应的空间，供系统中当前运行的程序使用。
- 等到某个程序要使用 SWAP 中的数据时，系统会再次从 SWAP 中读取之前保存的数据，并写回到物理内存中。

如果你细心的话，无论是使用 Linux 还是 Windows，当切换到一个长久没打开的软件窗口时，硬盘灯会呼呼地闪，这极有可能就是系统正在将“交换分区/虚拟内存”中的数据重新写回物理内存中的过程。是不是感觉一下子更了解你的电脑了呢？

SWAP 设置为多大比较合适呢

谣言是无处不在的，在 Linux 的世界里，也有不少以讹传讹的谣言，譬如说：

> SWAP大小应该设置成物理内存大小的2倍。如果物理内存是4GB，那么请将你的SWAP设置成8GB。

那么，到底应该把 SWAP 设置成多大，才合适呢？这个问题，估计很难有人能说清楚。

我们在这里会给大家一些建议的策略和方法：

- 对于普通的家用系统来说，随着内存的不断扩大，内存几乎已经足够使用了，SWAP 已经很少会派上用场，所以家用电脑完全可以不设置 SWAP 空间。

- 对于服务器来说，SWAP 还是非常有必要的，毕竟越来越复杂的程序，会导致内存大量的消耗。在内存不足时，SWAP 可以解燃眉之急，不至于出现内存耗尽服务器宕掉的情况。
- 对于服务器来说，SWAP 建议设置为内存的 1～2.5 倍之间的数值，可以防止内存耗尽的窘境。

如何查看 SWAP 分区大小呢

非常简单，只需通过 free 命令就可以查看到了。

```
[roc@roclinux ~]$ free -m
             total       used       free     shared    buffers     cached
Mem:          3950       2262       1687          0        407        952
-/+ buffers/cache:        903       3047
Swap:         1953          0       1953
```

从输出内容中可以看出，最后一行显示的便是 SWAP 信息。这台服务器的 SWAP 大小是 1953MB，目前还处在完全空闲的状态，没有被系统使用。

而有些服务器的输出，则是下面这个样子的：

```
[roc@roclinux ~]$ free -m
             total       used       free     shared    buffers     cached
Mem:          3031       1833       1197          0        283        421
-/+ buffers/cache:       1127       1903
Swap:            0          0          0
```

Swap 那一行全部是 0，这说明这台服务器没有开启 SWAP。那么如何开启和管理 SWAP 呢？我们接着往下看。

什么媒介可以作为 SWAP 呢

首先，要强调的一点是，SWAP 本质上就是硬盘的一块区域而已，用来存储内存中临时要交换的数据，所以大家不要把 SWAP 看得过于神秘。

那么问题来了，SWAP 到底是硬盘上怎样的一块区域呢？其实很简单，Linux 给了我们两种选择：

- 可以是一个独立的分区：这个很好理解，这个独立的区域，就用来作为交换分区使用。
- 可以是一个独立的文件：这个比较新颖，也比较灵活，内存中要交换的数据就会存到这个文件里。

纸上得来终觉浅，我们现在就开始建立属于自己的 SWAP 吧！

独立分区型 SWAP——创建

我们的服务器有一个独立的分区/dev/sda2，这次我们就把它做成 SWAP 分区。

首先要将/dev/sda2 分区更改为 SWAP 文件系统类型，于是，我们需要派出大将 fdisk：

```
#启动fdisk，处理/dev/sda2分区
 [root@roclinux ~]# fdisk /dev/sda2
Device contains neither a valid DOS partition table, nor Sun, SGI or OSF
disklabel
Building a new DOS disklabel with disk identifier 0xbba8bd03.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by
w(rite)

#输入p，表示要展示分区信息
Command (m for help): p

Disk /dev/sda: 40.0 GB, 40007761920 bytes
255 heads, 63 sectors/track, 4864 cylinders, total 78140160 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xc001c001

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048    40962047    20480000   83  Linux
/dev/sda2        40962048    41986047      512000   83  Linux
/dev/sda3        41986048    78139391    18076672   83  Linux

#输入t，表示要指定分区类型
Command (m for help): t

#输入2，表示要对第2个分区设置分区类型
Partition number (1-4): 2

#输入L，列出fdisk所支持的所有分区类型
Hex code (type L to list codes): L

 0  Empty           24  NEC DOS         81  Minix / old Lin bf  Solaris
 1  FAT12           27  Hidden NTFS Win 82  Linux swap / So c1  DRDOS/sec
(FAT-
 2  XENIX root      39  Plan 9          83  Linux           c4  DRDOS/sec
(FAT-
 3  XENIX usr       3c  PartitionMagic  84  OS/2 hidden C:  c6  DRDOS/sec
(FAT-
```

```
 4  FAT16 <32M      40  Venix 80286     85  Linux extended  c7  Syrinx
 5  Extended        41  PPC PReP Boot   86  NTFS volume set da  Non-FS data
 6  FAT16           42  SFS             87  NTFS volume set db  CP/M / CTOS
/ .
 7  HPFS/NTFS/exFAT 4d  QNX4.x          88  Linux plaintext de  Dell
Utility
 8  AIX             4e  QNX4.x 2nd part 8e  Linux LVM       df  BootIt
 9  AIX bootable    4f  QNX4.x 3rd part 93  Amoeba          e1  DOS access
 a  OS/2 Boot Manag 50  OnTrack DM      94  Amoeba BBT      e3  DOS R/O
 b  W95 FAT32       51  OnTrack DM6 Aux 9f  BSD/OS          e4  SpeedStor
 c  W95 FAT32 (LBA) 52  CP/M           a0  IBM Thinkpad hi eb  BeOS fs
 e  W95 FAT16 (LBA) 53  OnTrack DM6 Aux a5  FreeBSD         ee  GPT
 f  W95 Ext'd (LBA) 54  OnTrackDM6      a6  OpenBSD         ef  EFI
(FAT-12/16/
10  OPUS            55  EZ-Drive        a7  NeXTSTEP        f0
Linux/PA-RISC b
11  Hidden FAT12    56  Golden Bow      a8  Darwin UFS      f1  SpeedStor
12  Compaq diagnost 5c  Priam Edisk     a9  NetBSD          f4  SpeedStor
14  Hidden FAT16 <3 61  SpeedStor       ab  Darwin boot     f2  DOS
secondary
16  Hidden FAT16    63  GNU HURD or Sys af  HFS / HFS+      fb  VMware VMFS
17  Hidden HPFS/NTF 64  Novell Netware  b7  BSDI fs         fc  VMware
VMKCORE
18  AST SmartSleep  65  Novell Netware  b8  BSDI swap       fd  Linux raid
auto
1b  Hidden W95 FAT3 70  DiskSecure Mult bb  Boot Wizard hid fe  LANstep
1c  Hidden W95 FAT3 75  PC/IX           be  Solaris boot    ff  BBT
1e  Hidden W95 FAT1 80  Old Minix

#输入82，即指定分区类型为swap
Hex code (type L to list codes): 82
Changed system type of partition 2 to 82 (Linux swap / Solaris)

Command (m for help): p

Disk /dev/sda: 40.0 GB, 40007761920 bytes
255 heads, 63 sectors/track, 4864 cylinders, total 78140160 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xc001c001

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048    40962047    20480000   83  Linux
/dev/sda2         40962048    41986047      512000   82  Linux swap /
Solaris
/dev/sda3       41986048    78139391    18076672   83  Linux

#再次输入p，查看最新分区信息
Command (m for help):w
[root@roclinux ~]# partprobe
[root@roclinux ~]# mkswap /dev/sda2
Setting up swapspace version 1, size = 511996 KiB
```

独立分区型 SWAP——启用

我们再看一下当前的情况：

```
[root@roclinux ~]# free -m
             total       used       free     shared    buffers     cached
Mem:           992        568        424          0         31        386
-/+ buffers/cache:        151        841
Swap:            0          0          0
```

准备好，我们现在就要启用人生中的第一个 SWAP 啦：

```
[root@roclinux ~]# swapon /dev/sda2
[root@roclinux ~]# swapon -s
Filename                        Type            Size    Used    Priority
/dev/sda2                               partition   511996  0   -1
[root@roclinux ~]# free -m
             total       used       free     shared    buffers     cached
Mem:           992        569        423          0         31        386
-/+ buffers/cache:        151        841
Swap:          499          0        499
```

看！SWAP 已经正式入驻我们的磁盘了。

独立分区型 SWAP——关闭

虽然俗话说“请神容易送神难”，但是在 Linux 的世界里，送走 SWAP 也是件很容易的事情，使用 swapoff 就可以轻松搞定。

```
[root@roclinux ~]# swapoff /dev/sda2
[root@roclinux ~]# swapon -s
Filename                        Type            Size    Used    Priority
[root@roclinux ~]# free -m
             total       used       free     shared    buffers     cached
Mem:           992        588        404          0         32        398
-/+ buffers/cache:        157        835
Swap:            0          0          0
```

独立文件型 SWAP——创建

这是另外一种 SWAP 媒介，即通过我们指定的一个文件来作为 SWAP。

这种方式既新颖又灵活，在服务器分区已经不易改动的情况下，大家可以使用这种解决方案。

假如我们要创建一个 256MB 的 SWAP 空间，那么就要先创建一个 256MB 大小的文件，使用 dd 命令最为方便：

```
[root@roclinux tmp]# dd if=/dev/zero of=/tmp/rocswap bs=1M count=256
256+0 records in
256+0 records out
268435456 bytes (268 MB) copied, 12.6166 s, 21.3 MB/s
[root@roclinux tmp]# ls -hl rocswap
-rw-r--r-- 1 root root 256M Jan  9 12:24 rocswap
```

看，我们创建了一个正好 256MB 的文件 rocswap，再下面的步骤和独立分区型 SWAP 的创建方法就完全一致了：

```
[root@roclinux tmp]# mkswap /tmp/rocswap
Setting up swapspace version 1, size = 262140 KiB
no label, UUID=b724362f-887e-4718-b22d-c187432817a4
```

独立文件型 SWAP——启用

```
[root@roclinux tmp]# swapon /tmp/rocswap
[root@roclinux tmp]# swapon -s
Filename                Type       Size    Used    Priority
/tmp/rocswap                            file       262140  0   -1
[root@roclinux tmp]# free -m
             total       used       free     shared    buffers     cached
Mem:           992        914         78          0         17        637
-/+ buffers/cache:        259        733
Swap:          255          0        255
```

独立文件型 SWAP——关闭

仍然是使用 swapoff 命令来实现：

```
[root@roclinux tmp]# swapoff /tmp/rocswap
[root@roclinux tmp]# free -m
             total       used       free     shared    buffers     cached
Mem:           992        914         78          0         17        637
-/+ buffers/cache:        259        733
Swap:            0          0          0
```

到底如何定义“内存不足”呢？

还记得前面卖的一个关子吗，那段话是这样说的：

> 当Linux系统发现物理内存使用量不足时（那到底如何定义“不足”呢，留个悬念，后面揭晓），就会选择内存中较长时间没有被访问和更新的内存数据，将这些内存数据临时写到SWAP中，并释放内存中相应的空间，供系统中当前运行的程序使用。

问题来了，到底怎么定义“内存不足”呢？这个问题如果在面试时提出来，将会是一个很难很难的问题，会难住 95%以上的同学！

我们现在就揭晓答案啦！

其实，在 Linux 系统中，有一个参数会来辅助控制“内存不足”的界限，它就是 swappiness：

```
[root@roclinux tmp]# cat /proc/sys/vm/swappiness
60
```

swappniess 可以设置的范围是 0 到 100，当设置为不同值时，SWAP 策略的差异性如表 9 所示。

表 9 SWAP 策略的差异性

swappniess 值	SWAP 策略
vm.swappiness = 0	这会导致几乎禁用 SWAP，除非出现 Out of Memory 的情况
vm.swappiness = 1	这是除禁止 SWAP 之外的最最保守的 SWAP 策略
vm.swappiness = 10	通常在内存空间非常充足时，为了提升整体性能，会将值更改为 10，以便有效降低 SWAP 的频次
vm.swappiness = 60	默认值，属于中庸策略
vm.swappiness = 100	系统会极其激进地进行 SWAP，这将严重影响整体性能

在 Linux 内核代码中，触发 SWAP 的时机是由多个参数综合决定的，其中一些是不由用户控制的，因此我们只要知道 swappniess 可以辅助控制 SWAP 策略就足够了。

更加深入的、细节的 SWAP 策略和知识，我们留给感兴趣的 GEEK 们去研究啦。

好了，我们通过本文了解了 SWAP 的产生背景、作用以及建立的方法，还有在何种情况下会用到 SWAP。这无论是对于即将步入职场的同学们来说，还是 Linux 工程师们来说，都应该足够了。

4 vmstat 性能查看利器

序言

vmstat 这个命令真的非常实用，相信所有从事 Linux 相关工作的同学都使用过它。

不过，从我的观察和经验来看，很多同学只是使用了 vmstat 很初级的功能，也就是查看 CPU_IDLE、WAIT、磁盘 I/O 这几项。

今天，我们就和大家一起，走进这个低调的家伙，让大家知道它其实比你想象的要强大得多！

vmstat 的身世

vmstat，是一个 Linux 的性能监控工具，是 Virtual Memory Statistics 的缩写。从这个细节可以看出，这个命令最初只是被设计用来展示虚拟内存状态的，后来随着不断地迭代和扩展，现在变成了一个强大的性能监控工具。

vmstat，最初是在 1994 年被开发出来的，其作者是美国扬斯敦州立大学的 Henry Ware。貌似 Linux 的牛人都很低调，网上没有找到他更多的资料。

vmstat 的最常用用法

```
[roc@roclinux ~]$ vmstat 1 4
procs  -----------memory----------  ---swap--  -----io----  --system--
-----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0      0 1074612 280132 448120    0    0     0     2    1    2  6  0 93  0  0
 1  0      0 1074604 280132 448140    0    0     0     0  392  114  7  0 93  0  0
 1  0      0 1074684 280132 448140    0    0     0    12  424  123  8  1 91  0  0
 3  0      0 1074560 280136 448136    0    0     0   112  619  185 11  0 88  0  0
```

vmstat 最常用的方法是：

```
$ vmstat 输出间隔秒数 输出次数
```

从上面的例子可以看出，我们希望 vmstat 每隔 1 秒输出一次系统状态信息，共输出 4 次。

在了解了 vmstat 的最常用方法后，我们尝试用一张图，来帮助大家快速地解读 vmstat 的输出内容，如图 10 所示。

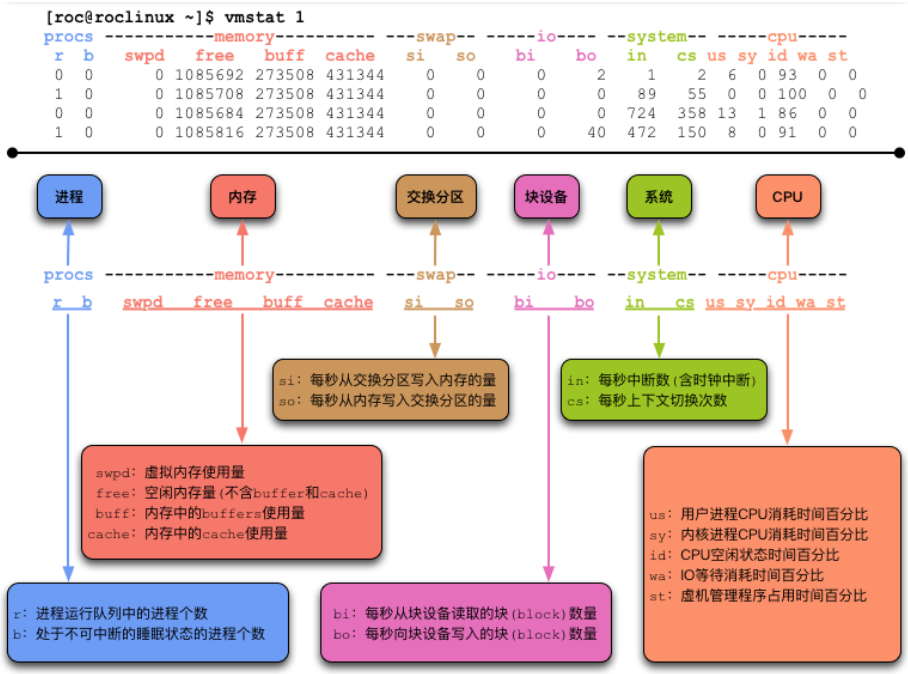


图 10　vmstat 的输出内容

通过图 10，相信大家可以一目了然地看出 vmstat 都输出了哪些内容，也一定能够找到自己想了解的性能信息了。

骗局！不要信第一行的数字

当服务器出异常时，我们总是心怀忐忑，想第一时间找到问题所在。

当我们运行 vmstat 1，并急于看到性能数据时，我们可能会陷入一个小小的骗局。

这个骗局，我们马上为你揭穿。

vmstat 输出的第一行数字，是自服务器启动至今的各项指标的平均值，而非最新状态值，从第二行开始的，才能反映服务器当前最新状态。这个小小的骗局曾让很

多运维工程师心惊肉跳过吧！

```
[roc@roclinux ~]$ vmstat 1
procs -----------------memory---------- ---swap-- -----io---- --system--
-----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0 1058412 276844 402680    0    0     0     2    0    0  6  0 93  0  0
（骗局就是这一行）
 0  0      0 1060252 276840 402684    0    0     0     0   85   60  0  0 100  0  0
（从这行以后，都是最新状态）
 0  0      0 1059436 276840 402556    0    0     0     0  707  342 13  1 87  0  0
```

-a 选项让你新学两个名词

不妨试试 vmstat 的-a 选项，看看输出会有什么变化？

```
[roc@roclinux ~]$ vmstat -a 1 4
procs  -----------memory----------  ---swap--  -----io----  --system--
-----cpu-----
 r  b   swpd   free  inact active   si   so    bi    bo   in   cs us sy id wa st
 0  0      0 1094412 386404 770400    0    0     0     2    1    0  6  0 93  0  0
 0  0      0 1094288 386404 770368    0    0     0     0   71   55  0  0 100  0  0
 0  0      0 1094288 386404 770512    0    0     0    56   64   74  0  0 97  3  0
 0  0      0 1094312 386404 770512    0    0     0     0  486  156  9  0 91  0  0
```

看出区别了嘛？这就是在 memory 信息中，原来的 buff 和 cache 信息消失了，取而代之的则是 inact 和 active，它们的全称则是：

- 非活跃内存（inactive memory）：一段时间内没有被使用过的内存。（如果你开启了 SWAP，那么这类内存会优先被交换到 SWAP 中）
- 活跃内存（active memory）：正在被使用中的内存。

看看你的服务器曾拥有过多少 forks

大家可以和周围从事 Linux 技术工作的同事或朋友比一比，看看谁的服务器曾拥有的 forks 最多。方法很简单，仍然使用 vmstat 来实现：

```
[roc@roclinux ~]$ vmstat -f
      465961 forks
[roc@roclinux ~]$ uptime
 21:54:33 up 73 days,  5:15,  1 user,  load average: 0.06, 0.16, 0.14
```

抛砖引玉，先看看我的服务器，持续运行了 73 天，在这 73 天里，曾累积拥有过 465961 个 forks！我知道这并不是很多，相信很多持续运行了一两年的服务器，这个 forks 数字会是个天文数字。

有些好奇的同学会问，到底什么是 forks 呢，是不是调用 fork()系统调用的总次数？

准确地说，forks 是由 fork、vfork 和 clone 三类系统调用（system calls）所产生的，可以理解为是系统创建的总任务（tasks）数。

又会有同学问了，任务数怎么理解呢？简单地说，一个进程可以包含一个或多个任务（tasks），而具体包括多少任务，则取决于此进程中创建了多少个线程。

有些运维工程师，会按照固定的时间间隔来提取这个数值的差值，以判断当前系统是不是因为 forks 骤增而导致了性能问题，大家也可以使用这种监控方法。

vmstat 其实是个多面手

vmstat 还可以用来查看 slab 信息、磁盘 I/O、分区 I/O，以及内存/系统事件信息等，我们通过表 10 来了解一下。

表 10 vmstat 用法

选　项	用　法	用法举例
-m	$ vmstat -m	展示内存 slabinfo
-s	$ vmstat -s	展示内存指标及系统事件信息
-d	$ vmstat -d	展示各磁盘的统计信息
-p	$ vmstat -p /dev/sda1	展示某一特定分区的 I/O 信息

我们在这里列出它们实际运行的样子，至于每一项内容的具体含义和背后意义，感兴趣的同学可以自己去深入研究。

```
#展示slabinfo
[roc@roclinux ~]$ vmstat -m
Cache                  Num  Total   Size  Pages
ccid3_hc_tx_sock         0      0    256     15
ccid3_hc_rx_sock         0      0    192     20
ccid2_hc_tx_sock         0      0   1216      3
ccid2_hc_rx_sock         0      0     16    202
tfrc_rxh_cache           0      0     32    112
tfrc_tx_hist             0      0     32    112
tfrc_li_hist             0      0     16    202
dccp_ackvec_record       0      0     64     59
dccp_ackvec              0      0    576      7
dccp_bind_bucket         0      0     64     59
（此处省略数百行）

#展示内存指标及系统事件信息
[roc@roclinux ~]$ vmstat -s
     3104152  total memory
     1947068  used memory
```

```
      725432  active memory
      764280  inactive memory
     1157084  free memory
      259728  buffer memory
      286416  swap cache
           0  total swap
           0  used swap
           0  free swap
   297496378 non-nice user cpu ticks
        6259 nice user cpu ticks
    14754274 system cpu ticks
   4907960040 idle cpu ticks
     1937617 IO-wait cpu ticks
      127763 IRQ cpu ticks
      772188 softirq cpu ticks
           0 stolen cpu ticks
    19786472 pages paged in
   112149388 pages paged out
           0 pages swapped in
           0 pages swapped out
   4149647789 interrupts
   1701093377 CPU context switches
   1447231547 boot time
     1146483 forks

#展示各磁盘的统计信息
[roc@roclinux ~]$ vmstat -d
disk-    ------------reads------------   ------------writes-----------
-----IO------
      total merged sectors      ms  total merged sectors      ms    cur
sec
ram0      0      0      0      0      0      0      0      0      0      0
ram1      0      0      0      0      0      0      0      0      0      0
ram2      0      0      0      0      0      0      0      0      0      0
ram3      0      0      0      0      0      0      0      0      0      0
ram4      0      0      0      0      0      0      0      0      0      0
ram5      0      0      0      0      0      0      0      0      0      0
ram6      0      0      0      0      0      0      0      0      0      0
ram7      0      0      0      0      0      0      0      0      0      0
ram8      0      0      0      0      0      0      0      0      0      0
ram9      0      0      0      0      0      0      0      0      0      0
ram10     0      0      0      0      0      0      0      0      0      0
ram11     0      0      0      0      0      0      0      0      0      0
ram12     0      0      0      0      0      0      0      0      0      0
ram13     0      0      0      0      0      0      0      0      0      0
ram14     0      0      0      0      0      0      0      0      0      0
ram15     0      0      0      0      0      0      0      0      0      0
loop0     0      0      0      0      0      0      0      0      0      0
loop1     0      0      0      0      0      0      0      0      0      0
loop2     0      0      0      0      0      0      0      0      0      0
loop3     0      0      0      0      0      0      0      0      0      0
loop4     0      0      0      0      0      0      0      0      0      0
loop5     0      0      0      0      0      0      0      0      0      0
loop6     0      0      0      0      0      0      0      0      0      0
```

```
loop7     0     0      0      0     0     0      0      0     0     0
sda   145633  63585 6140210  814395 4756224 10230226 119811328 19379273
0  11432
sdb   353646  79386 33430302 1779191 6900332 6162257 104487664 30445096
0  15557

#展示某一特定分区的I/O信息
[roc@roclinux ~]$ vmstat -p /dev/sda1
sda1         reads   read sectors  writes    requested writes
            145470    6138906    4746154  119811544
```

怎么算正常，怎么算有问题呢

终于说到最重要的一个段落了，这也是各位同学最想学习和掌握的内容，那就是到底怎样从 vmstat 中看出服务器的问题呢？

这一段其实真的不好写，因为不同的服务器、不同的操作系统、不同的应用场景，都各有特点，不能简单地通过一个阈值来区分“正常”和“异常”。所以我们更多地使用了“较大”、“较小”、“很低”、“很高”等形容词，也是因为很难给出一个具体的参考值。

所以，这里要善意地提醒大家，这个段落更多的是根据经验给出的建议，如果和你的场景违背，还请见谅。

下面我们就基于 vmstat 的输出来给出一些建议吧。

```
[roc@roclinux ~]$ vmstat 1 4
procs  -----------memory----------  ---swap--  -----io----  --system--
-----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0      0 1074612 280132 448120    0    0     0     2    1    2  6  0 93  0  0
 1  0      0 1074604 280132 448140    0    0     0     0  392  114  7  0 93  0  0
 1  0      0 1074684 280132 448140    0    0     0    12  424  123  8  1 91  0  0
 3  0      0 1074560 280136 448136    0    0     0   112  619  185 11  0 88  0  0
```

1. 如果 cache 的数值较大，则说明系统缓存了较多的磁盘数据，利于磁盘 I/O 性能的提升。这个时候，往往 bi 会相对较小，因为很多读磁盘的操作都由 cache 来承担了。
2. 而 si 和 so 则是读写 SWAP 的量，这两个值如果长期大于 0，则表示系统需要经常读写交换分区，这会很消耗 CPU 资源和磁盘 I/O 性能。这时就要格外关注了，如果确定是系统的物理内存存在瓶颈，那么就要通过扩容或服务迁移来解决问题。

3．如果 free 的数值很低，甚至接近 0 了，也不一定就是系统内存快耗尽了。要同时看 buff 和 cache 的量，大部分情况是 buff 和 cache 占用了很多内存资源，这反而是好事，说明系统把空闲的内存都利用起来作为缓存，提升系统 I/O 性能了。而当系统真正需要内存时，buff 和 cache 是可以随时被系统征调回来的。

4．如果发现 bi 和 bo 的值很大，则说明系统正在进行大量的磁盘读写操作。如果是符合预期的还好，如果不是的话，就要去查一查到底是哪块磁盘、哪个分区在进行大量读写。

5．如果 us 的数值经常大于 50%，则说明用户进程所占用的 CPU 时间较多，这或许说明所开发的程序需要进行一定程度的性能优化了。

6．而 sy 是内核所消耗的 CPU 时间，这个数值不应该很高。如果很高，则一定是系统哪里出了问题。

7．如果 wa 较高，则说明 CPU 总是在等待 I/O 操作。这表明磁盘已经成为主要瓶颈，我们可以把磁盘升级为高性能磁盘，也可以查一下我们的程序是不是存在大量的随机读操作，如果是的话，可以考虑调整为顺序读或者考虑增加读缓存。

8．而 r 表示的是正在运行队列中的任务数，如果这个数值总是超过服务器的 CPU 核数，则说明 CPU 已经成为性能瓶颈，可以考虑开启超线程、更换更多核的 CPU、调整某些进程的 NICE 优先级等措施。

9．如果你正在运行一个视频编解码的任务，那么 us 的数值可能会很高，甚至达到 95%以上，这是符合预期的，不要紧张。如果你要运行一个能产生大量随机数的程序，或者其他包含系统调用的程序，那么 sy 可能会非常高，这也很正常。

10．假如你打开了一个大型软件，比如 Office 程序，然后过了一段时间，你又希望启动另一个大型游戏，这时候，你会发现 so 的数值会升高，这表示系统正在将内存里的闲置数据写入到 SWAP 中，这也是正常现象。如果你希望减少这类 SWAP 写入，那么就去加大内存吧。

5 mpstat，让你了解 CPU 的心

带我走进多核的内心

mpstat，全称是 multiprocessor statistics，正如它的名字一样，它所擅长的技能正是多处理器的统计工作。

让我们来看看它的常见用法：

```
[roc@roclinux splan]$ mpstat -P ALL 1
Linux 2.6.32-220.4.1.el6.x86_64 (roclinux)     2016年02月03日  _x86_64_      (4 CPU)

16时48分16秒  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest   %idle
16时48分17秒  all    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
16时48分17秒    0    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
16时48分17秒    1    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
16时48分17秒    2    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
16时48分17秒    3    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00

16时48分17秒  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest   %idle
16时48分18秒  all    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
16时48分18秒    0    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
16时48分18秒    1    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
16时48分18秒    2    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
16时48分18秒    3    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
```

发现了么，我们使用了–P ALL 这样的选项和参数，其作用是展示出每一个 CPU 核的运行状态。

我们的这台服务器总共有 4 个 CPU 核，mpstat 清晰地展示出了综合状态，以及每一个 CPU 核的运行状态。

而命令中最后的参数 1，则表示要求 mpstat 每隔 1 秒钟，产出一组新的状态数据。如果你使用过 vmstat 的话，你会很熟悉这种用法。

如果我们只想关注第 1 个 CPU 核（编号为 0）的话，应该怎么办呢？其实很简单，只需要把 ALL 改成 0，就可以了。

```
[roc@roclinux splan]$ mpstat -P 0 1
Linux 2.6.32-220.4.1.el6.x86_64 (roclinux)     2016年02月03日  _x86_64_      (4 CPU)

17时25分34秒  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest   %idle
17时25分35秒    0    2.30    0.00    0.00    1.15    0.00    0.00    0.00    0.00   96.55
17时25分36秒    0    3.16    0.00    0.00    0.00    0.00    0.00    0.00    0.00   96.84
17时25分37秒    0   35.59    0.00    1.69    0.00    0.00    0.00    0.00    0.00   62.71
17时25分38秒    0   18.29    0.00    1.22    0.00    0.00    0.00    0.00    0.00   80.49
```

看到了吧，现在 mpstat 每隔 1 秒钟只会输出编号为 0 的 CPU 核的状态数据了。

CPU 的各类指标含义

如果你读过 vmstat 命令的文章的话，那么应该对 CPU 的这几个指标相当熟悉了。所以建议那些没有看过那篇文章的小伙伴们，赶快去阅读一下。

在这里，我们对 CPU 指标做一个基本的介绍，如表 11 所示。

表 11　CPU 指标

指　　标	说　　明
%user	用户进程所使用 CPU 的百分比
%nice	对进程进行降级时 CPU 的百分比
%sys	内核进程使用的 CPU 百分比
%iowait	等待进行 I/O 所使用的 CPU 时间百分比
%irq	处理系统中断的 CPU 百分比
%soft	软件中断的 CPU 百分比
%steal	虚机管理程序占用的 CPU 百分比
%guest	运行虚拟处理器占用的 CPU 百分比
%idle	CPU 的空闲时间

mpstat 的作者

Linux 界的很多名人都是很低调的，低调到在互联网上都很难搜索到他们的任何一张照片。

mpstat 的作者的名字是 Sebastien Godard，中文是塞巴斯蒂安·高达，他来自法国巴黎，虽然从照片来看已然是个大叔了，但戴上墨镜还是很酷的。

这位大叔可不是一般人，sysstat 工具套装全部是他的杰作。什么？你不知道 sysstat 是什么。那如果我说 mpstat、sar、iostat、pidstat 都是这个工具套装里的工具，你应该会对这位大叔唏嘘不已了吧。

回顾中断的知识

因为接下来就要介绍 mpstat 的 CPU 中断统计功能了，所以呢，趁这个机会，先和大家回顾一下有关中断的知识。

处理器的运算速度一般要比外部硬件快很多。以读取硬盘为例，如果是简单的顺序执行，则 CPU 必须等待很长时间，不停地轮询硬盘是否读取完毕，这会浪费很多 CPU 时间。中断提供了这样一种机制，使得读取硬盘这样的操作可以交给硬件来完成，CPU 挂起当前进程，将控制权转交给其他进程，待硬件处理完毕后通知 CPU，操作系统把当前进程设为活动的，从而允许该进程继续执行，处理读取硬盘的结果。

另一方面，有些事件不是程序本身可预见的，需要硬件以某种方式告诉进程。例如，时钟中断为定时器提供了基础，如果没有时钟中断，那么程序只能每执行几条指令就检查一下当前系统时间，这在效率上是不可接受的。

说说 I 选项

-I 选项，代表的是 Interrupts，这个选项用来报告 CPU 中断处理情况。它有三个可选参数：

- SUM
- CPU
- ALL

第一个是参数 SUM：会展示所有 CPU 的中断数之和。

```
[roc@roclinux 20160202]$ mpstat -I SUM 1
Linux 2.6.32-220.4.1.el6.x86_64 (roclinux)     2016年02月04日  _x86_64_
(4 CPU)

15时48分47秒  CPU    intr/s
15时48分48秒  all     75.00
```

```
15时48分49秒  all    64.00
```

第二个是参数 CPU：会展示每一个 CPU 的中断处理情况，但不会展示所有 CPU 的总和数据。

```
[roc@roclinux 20160202]$ mpstat -I CPU 1
Linux 2.6.32-220.4.1.el6.x86_64 (roclinux)     2016年02月04日  _x86_64_       (4
CPU)

15时49分50秒  CPU       0/s       1/s       7/s       8/s       9/s      12/s
NMI/s      LOC/s      SPU/s      PMI/s      PND/s      RES/s      CAL/s      TLB/s
TRM/s      THR/s      MCE/s      MCP/s      ERR/s      MIS/s
15时49分51秒    0      0.00      0.00     12.87      0.00      0.00      0.00
0.00      34.65      0.00      0.00      0.00      0.00      0.00      0.00
0.00       0.00      0.00      0.00      0.00      0.00
15时49分51秒    1      0.00      0.00      0.00      0.00      0.00      0.00
0.00      15.84      0.00      0.00      0.00      0.00      0.00      0.00
0.00       0.00      0.00      0.00      0.00      0.00
15时49分51秒    2      0.00      0.00      0.00      0.00      0.00      0.00
0.00       2.97      0.00      0.00      0.00      0.99      0.00      0.00
0.00       0.00      0.00      0.00      0.00      0.00
15时49分51秒    3      0.00      0.00      0.00      0.00      0.00      0.00
0.00       8.91      0.00      0.00      0.00      0.99      0.00      0.00
0.00       0.00      0.00      0.00      0.00      0.00
```

第三个是参数 ALL，很好理解，就是把 SUM 和 CPU 的内容合在一起展示。

```
[roc@roclinux 20160202]$ mpstat -I ALL 1
Linux 2.6.32-220.4.1.el6.x86_64 (roclinux)     2016年02月04日  _x86_64_       (4
CPU)

16时55分43秒  CPU    intr/s
16时55分44秒  all    82.00

16时55分43秒  CPU       0/s       1/s       7/s       8/s       9/s      12/s
NMI/s      LOC/s      SPU/s      PMI/s      PND/s      RES/s      CAL/s      TLB/s
TRM/s      THR/s      MCE/s      MCP/s      ERR/s      MIS/s
16时55分44秒    0      0.00      0.00     17.00      0.00      0.00      0.00
0.00      38.00      0.00      0.00      0.00      2.00      0.00      0.00
0.00       0.00      0.00      0.00      0.00      0.00
16时55分44秒    1      0.00      0.00      0.00      0.00      0.00      0.00
0.00       8.00      0.00      0.00      0.00      1.00      0.00      0.00
0.00       0.00      0.00      0.00      0.00      0.00
16时55分44秒    2      0.00      0.00      0.00      0.00      0.00      0.00
0.00      11.00      0.00      0.00      0.00      0.00      0.00      0.00
0.00       0.00      0.00      0.00      0.00      0.00
16时55分44秒    3      0.00      0.00      0.00      0.00      0.00      0.00
0.00       5.00      0.00      0.00      0.00      0.00      0.00      0.00
0.00       0.00      0.00      0.00      0.00      0.00
```

好了，有了 mpstat 神器，我们就可以关注到每一个 CPU 核的运行状况了。CPU 尽在掌握的感觉，是不是很爽呢！

6 top 命令庖丁解牛之一——入门

如果说 vmstat 还有些同学不知道的话，那么 top 命令应该是大家最耳熟能详的 Linux 命令之一了吧。

下面我们就来和大家聊聊 top 这个命令。闲言少叙，我们直接进入正题。

top 是做什么的呢

top 命令，是非常常用的 Linux 性能监控工具，运行时，它会独占屏幕，并周期性地更新，让工程师们可以快速了解服务器的运行状态。

通过 top，我们可以了解到服务器的 CPU 负载情况、内存状态、SWAP 使用状况，以及详尽的进程级运行状态，可谓应有尽有。

我们最常使用的场景，大概有以下几个：

1. 找到是哪些淘气的进程在大量消耗 CPU 资源。
2. 看看哪些进程消耗了宝贵的内存空间。
3. 看看哪些进程占用的 CPU 时间最多。

除了这些场景，top 还有一些不为人知的便捷功能，接下来我们就为大家一一介绍。

说说 top 的历史

如果大家 man top 的话，会看到 HISTORY Former top 这样一个段落，这就是来专门介绍 top 命令发展历史的。

在其他 Linux 命令中，是很少见到有历史介绍的，这也说明了 top 命令是个有故事的命令，值得我们去了解一下。

在这里，我就按照官方原文，简单地为大家介绍一下 top 的历史：

- 最早期的 top 命令，是基于 ps 命令来设计和开发的，作者是 Roger Binns 先生。
- 后来，Robert Nation 先生将 top 进行了重构，以便能够支持 proc 文件系统标准。
- 接下来，Helmut Geyer 先生在此基础上增加了“可配置域”的功能，让 top 更加灵活强大。
- 而目前大家在用的 top 命令，则是由 James C. Warner 来整体重写的，在此过程中得到了 Albert D. Cahalan 和 Craig Small 的很大帮助。
- 现在 top 命令的维护者是 Albert D. Cahalan 先生。

可见，参与了 top 设计和开发的人有好几位，在这个过程中，倾注了他们大量的智慧和经验，正因如此，才造就了 top 现在“最常用 Linux 性能监控工具”的地位。

最简单的 top 用法

擒贼就要先擒王，学命令则要先学最简单的用法。我们直接输入 top 就可以了：

```
[roc@roclinux ~]$ top
top - 14:52:33 up 75 days, 22:13,  1 user,  load average: 0.17, 0.15, 0.13
Tasks: 169 total,   1 running, 168 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,
0.0%st
Mem:   3104152k total,  1819224k used,  1284928k free,   292844k buffers
Swap:       0k total,       0k used,       0k free,   388040k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
    1 root      20   0 19224 1096  812 S  0.0  0.0   0:08.80 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      RT   0     0    0    0 S  0.0  0.0   0:00.78 migration/0
    4 root      20   0     0    0    0 S  0.0  0.0   0:04.30 ksoftirqd/0
    5 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    6 root      RT   0     0    0    0 S  0.0  0.0   0:00.06 watchdog/0
    7 root      RT   0     0    0    0 S  0.0  0.0   0:01.13 migration/1
    8 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/1
    9 root      20   0     0    0    0 S  0.0  0.0   0:02.87 ksoftirqd/1
   10 root      RT   0     0    0    0 S  0.0  0.0   0:00.04 watchdog/1
```

看，各类进程信息都已经呈现在你面前了，而且是周期性自动更新的，是不是很体贴。

如果想结束，按 q 键，就可以退出程序了。

top 命令输出内容——初探

top 命令的输出内容，简单看可以分为上下两部分，上半部分是服务器的系统级数据，下半部分则是服务器的进程级数据，如表 12 所示。

表 12 top 命令的输出内容

数　据	位置	内　容
系统级数据	上部	1．系统负载信息 2．CPU 信息 3．内存信息
进程级数据	下部	包含每一个进程的详细信息，即 PID、USER、PR、NI、VIRT、RES、SHR、S、%CPU、%MEM、TIME+、COMMAND

top 命令输出图如图 11 所示。

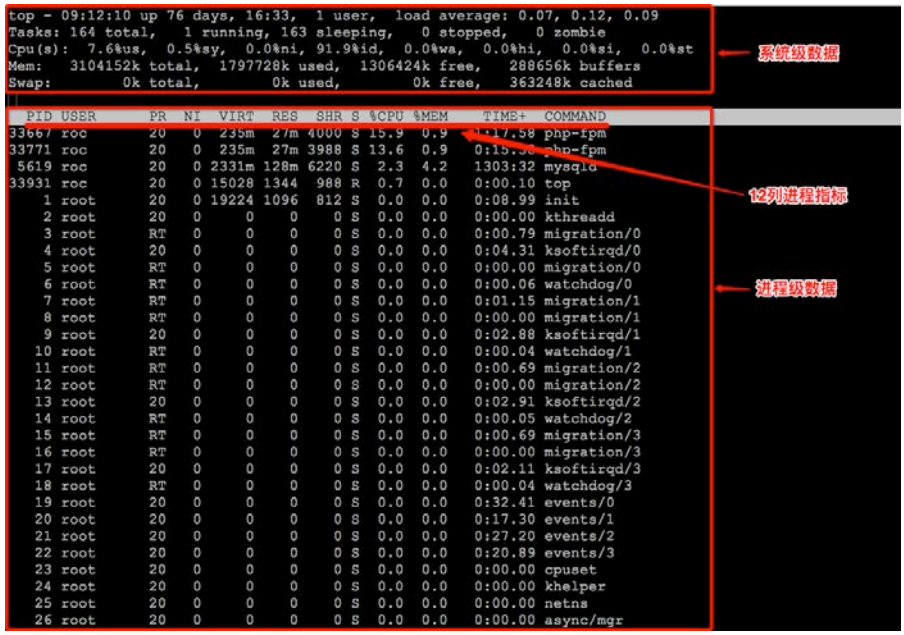


图 11 top 命令输出图

top 命令输出内容——系统级数据

我们从图 11 中已经看到了，top 命令输出中上半部分是系统级的数据，我们现在就来仔细看看它们都是什么吧！

老规矩，我们仍然看图说话，不同的是，我们会将其分解成五张图，以便让大家

一目了然，如图 12 至图 16 所示。

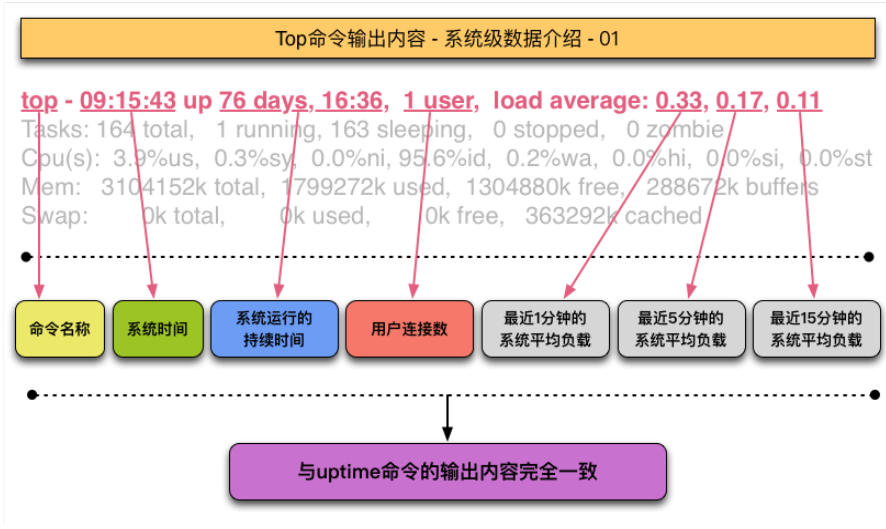


图 12　top 命令输出图（一）

图 13　top 命令输出图（二）

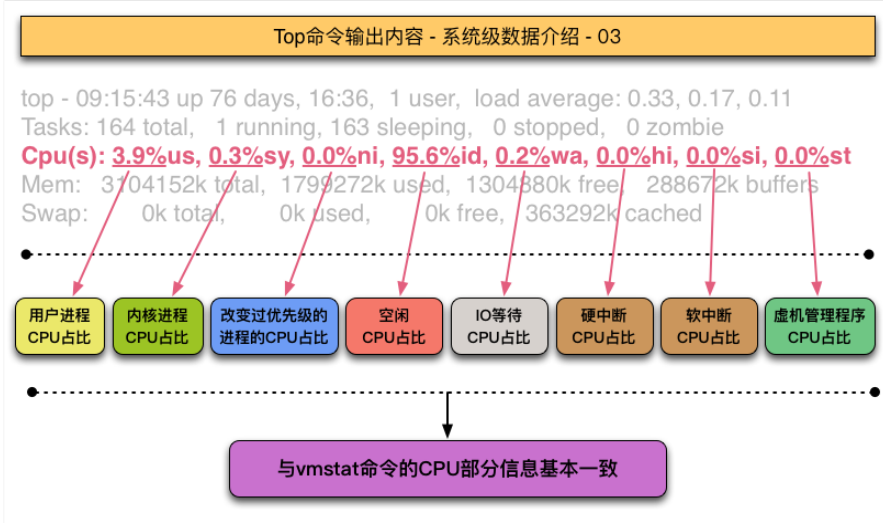


图 14 top 命令输出图（三）

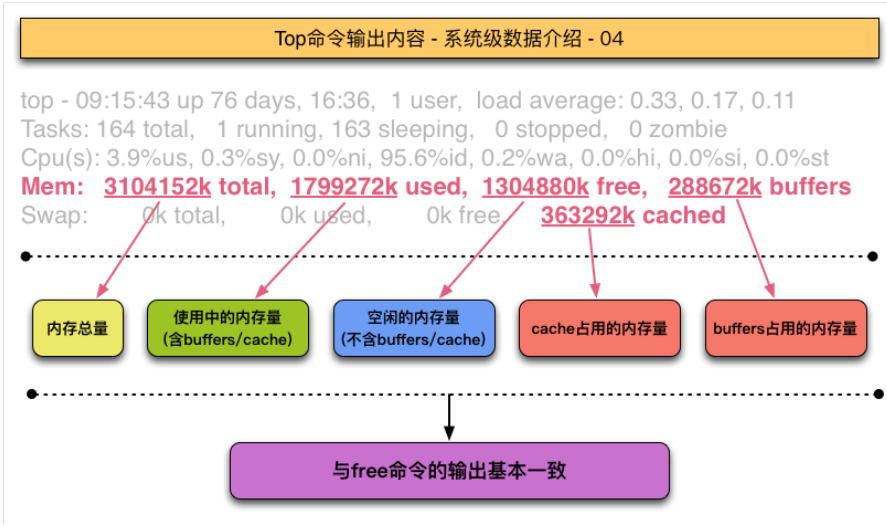


图 15 top 命令输出图（四）

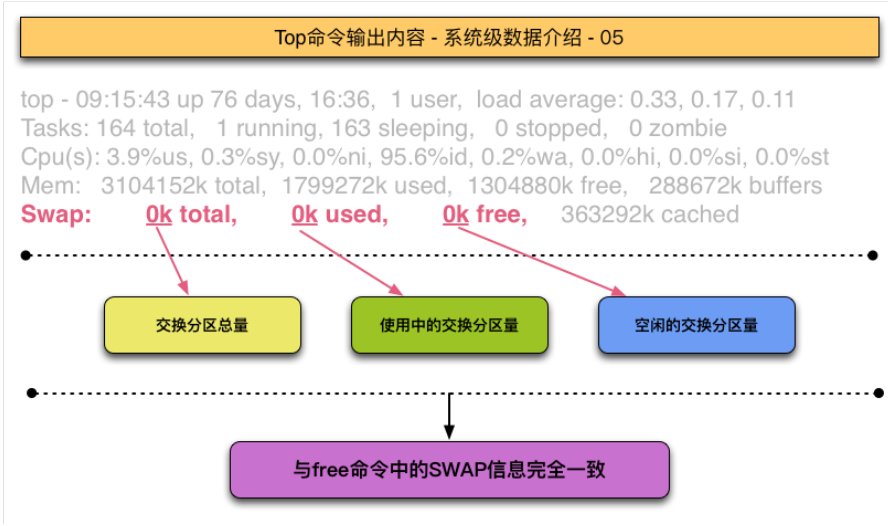


图 16　top 命令输出图（五）

好了，这个系列的第一篇文章，我们了解了 top 的发展历史，学习了 top 命令最简单的用法，也掌握了 top 命令系统级数据的含义。大家稍微休息一会儿，我们进入到这个系列的第二篇文章，有关 top 命令的列管理的话题。

7 top 命令庖丁解牛之二——列管理

教你两个小技巧 - f 和 o 的妙用

讲完系统级数据之后，我们应该进入“进程级数据”的内容啦，不过在此之前，还是要先介绍一个重要的知识，那就是“如何让你看到更丰富的进程信息”。

我们设置一个场景：通过 top 命令展示出进程的 PPID 信息，并且让 PPID 信息排在 PID 信息的后面。下面我们一起手把手来实现这个效果。

第一步：运行 top 命令，进入列选择页面。

单击快捷键 f，会进入进程信息列选择页面。

```
Current Fields:  AEHIOQTWKNMbcdfgjplrsuvyzX  for window 1:Def
Toggle fields via field letter, type any other key to return

* A: PID       = Process Id                                0x00000400
PF_SIGNALED
* E: USER      = User Name                                 0x00000800
PF_MEMALLOC
* H: PR        = Priority                                  0x00002000
PF_FREE_PAGES (2.5)
* I: NI        = Nice value                                0x00008000
debug flag (2.5)
* O: VIRT      = Virtual Image (kb)                        0x00024000
special threads (2.5)
* Q: RES       = Resident size (kb)                        0x001D0000
special states (2.5)
* T: SHR       = Shared Mem size (kb)                      0x00100000
PF_USEDFPU (thru 2.4)
* W: S         = Process Status
* K: %CPU      = CPU usage
* N: %MEM      = Memory usage (RES)
* M: TIME+     = CPU Time, hundredths
  b: PPID      = Parent Process Pid
  c: RUSER     = Real user name
  d: UID       = User Id
  f: GROUP     = Group Name
  g: TTY       = Controlling Tty
  j: P         = Last used cpu (SMP)
  p: SWAP      = Swapped size (kb)
```

```
  l: TIME      = CPU Time
  r: CODE      = Code size (kb)
  s: DATA      = Data+Stack size (kb)
  u: nFLT      = Page Fault count
  v: nDRT      = Dirty Pages count
  y: WCHAN     = Sleeping in Function
  z: Flags     = Task Flags
* X: COMMAND   = Command name/line
```

看，这里列出了 top 命令所能展示的进程的所有指标项，足有 26 个，正好对应 26 个字母！

第二步：添加所需列。

假如我们希望展示进程的 PPID 信息，那么我们就按照对应的字母提示，单击 b，此时，你会神奇地发现，原来 PPID 前面小写的字母 b，瞬间变成了大写，而且前面还多了一个星号（*），这就说明我们已经设置成功了。

第三步：回到 top 页。

我们按一下回车键，就可以回到 top 页了，可以看到 PPID 已经展示出来了！只不过它展示在了倒数第二列。

```
top - 14:29:19 up 76 days, 21:50,  1 user,  load average: 0.31, 0.23, 0.15
Tasks: 168 total,   1 running, 167 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.1%sy,  0.0%ni, 99.9%id,  0.0%wa,  0.0%hi,  0.0%si,
0.0%st
Mem:   3104152k total,  1847248k used,  1256904k free,   290664k buffers
Swap:       0k total,        0k used,        0k free,   370428k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+   PPID COMMAND
    1 root      20   0 19224 1096  812 S  0.0  0.0   0:08.99      0 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00      0 kthreadd
    3 root      RT   0     0    0    0 S  0.0  0.0   0:00.80      2
migration/0
    4 root      20   0     0    0    0 S  0.0  0.0   0:04.31      2
ksoftirqd/0
    5 root      RT   0     0    0    0 S  0.0  0.0   0:00.00      2
migration/0
    6 root     RT   0     0    0    0 S  0.0  0.0   0:00.06      2 watchdog/0
```

第四步：进入列顺序调整页。

PPID 和 PID 摆在一起，看起来才最方便，所以呢，我们就来学习怎样调整列的顺序。

单击英文字母 o，会进入一个和列选择页非常类似的页面，唯一的区别是上面的提示语有所不同。

```
Current Fields:  AEHIOQTWKNMBcdfgjplrsuvyzX  for window 1:Def
Upper case letter moves field left, lower case right

* A: PID       = Process Id                                    0x00000400
PF_SIGNALED
* E: USER      = User Name                                     0x00000800
PF_MEMALLOC
* H: PR        = Priority                                      0x00002000
PF_FREE_PAGES (2.5)
* I: NI        = Nice value                                    0x00008000
debug flag (2.5)
* O: VIRT      = Virtual Image (kb)                            0x00024000
special threads (2.5)
* Q: RES       = Resident size (kb)                            0x001D0000
special states (2.5)
* T: SHR       = Shared Mem size (kb)                          0x00100000
PF_USEDFPU (thru 2.4)
* W: S         = Process Status
* K: %CPU      = CPU usage
* N: %MEM      = Memory usage (RES)
* M: TIME+     = CPU Time, hundredths
* B: PPID      = Parent Process Pid
  c: RUSER     = Real user name
  d: UID       = User Id
  f: GROUP     = Group Name
  g: TTY       = Controlling Tty
  j: P         = Last used cpu (SMP)
  p: SWAP      = Swapped size (kb)
  l: TIME      = CPU Time
  r: CODE      = Code size (kb)
  s: DATA      = Data+Stack size (kb)
  u: nFLT      = Page Fault count
  v: nDRT      = Dirty Pages count
  y: WCHAN     = Sleeping in Function
  z: Flags     = Task Flags
* X: COMMAND   = Command name/line
```

第五步：调整 PPID 列的顺序。

这个页面的玩法，和刚才的列选择页有些不同，在这个页面里，当你键入对应字母的小写字母时，则相应的列会下移（在 top 页面中其实就是相应的列向右移）；当你键入对应字母的大写字母时，则相应的列会上移（在 top 页面中其实就是相应的列向左移）。

举个例子，我们希望将 PPID 列移到 PID 的右边来，那么我就连续地按大写字母 B，直到变成下面这个样子。

```
Current Fields:  ABEHIOQTWKNMcdfgjplrsuvyzX  for window 1:Def
Upper case letter moves field left, lower case right

* A: PID       = Process Id                                    0x00000400
PF_SIGNALED
```

```
* B: PPID      = Parent Process Pid                          0x00000800
PF_MEMALLOC
* E: USER      = User Nameocess Pid                          0x00002000
PF_FREE_PAGES (2.5)
* H: PR        = Priorityrocess Pid                          0x00008000
debug flag (2.5)
* I: NI        = Nice valuecess Pid                          0x00024000
special threads (2.5)
* O: VIRT      = Virtual Image (kb)                          0x001D0000
special states (2.5)
* Q: RES       = Resident size (kb)b)                        0x00100000
PF_USEDFPU (thru 2.4)
* T: SHR       = Shared Mem size (kb)
* W: S         = Process Status Pid
* K: %CPU      = CPU usageocess Pid
* N: %MEM      = Memory usage (RES)hs
* M: TIME+     = CPU Time, hundredths
  c: RUSER     = Real user name
  d: UID       = User Ider name
  f: GROUP     = Group Name
  g: TTY       = Controlling Tty
  j: P         = Last used cpu (SMP)
  p: SWAP      = Swapped size (kb)
  l: TIME      = CPU Time
  r: CODE      = Code size (kb)
  s: DATA      = Data+Stack size (kb)
  u: nFLT      = Page Fault count
  v: nDRT      = Dirty Pages count
  y: WCHAN     = Sleeping in Function
  z: Flags     = Task Flags
* X: COMMAND   = Command name/line
```

第六步：回到 top 页。

按一下回车键，则回到了 top 页，这时，PPID 已经如我们所愿，排到 PID 的后面了，看起来是不是舒服多了。

```
top - 14:41:26 up 76 days, 22:02,  1 user,  load average: 0.10, 0.10, 0.09
Tasks: 168 total,   1 running, 167 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.1%sy,  0.0%ni, 99.9%id,  0.0%wa,  0.0%hi,  0.0%si,
0.0%st
Mem:   3104152k total,  1834500k used,  1269652k free,   290684k buffers
Swap:        0k total,        0k used,        0k free,   370288k cached

  PID  PPID USER     PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
36547 36193 roc    20   0 15028 1352  992 R  0.3  0.0   0:01.05 top
    1    0 root     20   0 19224 1096  812 S  0.0  0.0   0:08.99 init
    2    0 root     20   0    0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3    2 root     RT   0    0    0    0 S  0.0  0.0   0:00.80 migration/0
    4    2 root     20   0    0    0    0 S  0.0  0.0   0:04.31 ksoftirqd/0
    5    2 root     RT   0    0    0    0 S  0.0  0.0   0:00.00 migration/0
    6    2 root     RT   0    0    0    0 S  0.0  0.0   0:00.06 watchdog/0
    7    2 root     RT   0    0    0    0 S  0.0  0.0   0:01.17 migration/1
    8    2 root     RT   0    0    0    0 S  0.0  0.0   0:00.00 migration/1
```

```
    9     2 root      20   0     0    0    0 S  0.0  0.0   0:02.88 ksoftirqd/1
```

将精心配置的页面方案保留下来

当你精心调整了 top 的各种列和它们的顺序之后，你一定希望以后再打开 top 的时候，也是同样的页面，因为这是属于你的个人定制版。

方法也很简单，就是使用快捷键 W（大写字母 w），当你按下 W 的时候，top 就会把当前的配置保存到你的家目录的~/.toprc 文件中去了。

好了，我们已经把增加列和调整列的方法都学会了，下面，我们就进入 top 命令的进程级数据的讲解了。

8 top 命令庖丁解牛之三——进程数据

top 命令输出内容——进程级数据

利用上面教的小技巧，我们把进程的所有 26 个指标项都展示了出来，截图还是相当壮观的，如图 17 所示。

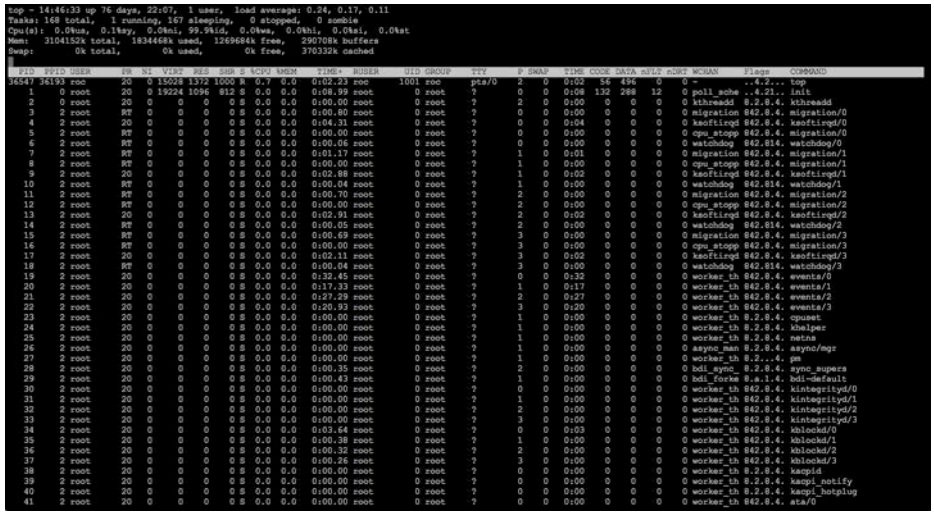


图 17　26 个指标项

然后我们通过表 13 这张巨大的表格，为大家逐一解释这 26 个指标的含义。

表 13　26 个指标的含义

指　　标	说　　明	参考值
PID	进程 ID	
PPID	进程的父进程 ID	
USER	进程拥有者的用户名，官方也叫 effective user，也就是启动进程时所用的用户名	
PR	进程的优先级	

续表

指　　标	说　　明	参考值
NI	进程的 NICE 值，用于调节优先级	–20 到 20 之间的整数。负数表示优先级提升，正数表示优先级降低
VIRT	进程使用的虚拟内存量	
RES	进程使用的且未被换出的物理内存量	
SHR	共享内存量	
S	进程所处的状态	R：运行 S：睡眠 D：不可中断的睡眠状态 T：跟踪、停止 Z：僵尸
%CPU	进程所占用的 CPU 时间占比	
%MEM	进程所使用的物理内存占比	
TIME+	进程所占用的 CPU 时间总和，单位为 1/100 秒	
RUSER	进程拥有者的实际用户名，官方也叫 real user，也就是登录到 Shell 时的用户名	
UID	进程所有者的用户 ID，及 USER 所对应的用户 ID	
GROUP	进程所有者的用户组名称	
TTY	启动此进程的终端名称	若对应终端，则显示问号（?）
P	此进程最近一个时刻所使用的 CPU 编号	
SWAP	进程使用的且已被换出的虚拟内存量	
TIME	进程所占用的 CPU 时间总和，单位为秒	
CODE	进程对应的可执行代码所占用的物理内存量	
DATA	进程对应的数据部分（数据段、栈等）所占用的物理内存量	
nFLT	页面错误次数	
nDRT	从最后一次写入到此时此刻，被修改过的页面数	
WCHAN	如果此进程正在睡眠，则显示睡眠中的系统调用名	
Flags	进程标志	
COMMAND	进程所对应的命令名（可能含全路径）	

好了，这一篇的内容，就到这里了，内容不少，但能够真正把这 26 个指标了然于心，却并非易事。

9 top 命令庖丁解牛之四——排序大法

对进程级数据排序

在使用 top 时，我们是要经常进行排序的，比如查看是哪些进程在大量消耗 CPU，就需要按照 CPU 占用比例来对进程进行排序。

所以本篇文章，我们就专门为大家介绍 top 命令在排序方面的技巧。

如果你想知道是哪个进程占用了大量的 CPU 资源，那么就请使用快捷键 P（大写字母 p），它用来基于%CPU 进行全局排序，而且默认就是降序。是不是很简单？用这个方法，你就可以很快找到罪魁祸首了！

类似的，如果你想找到那个占用了你大量内存资源的进程，则可以使用快捷键 M，它可以按照%MEM 来进行全局降序排序。

除了 CPU、内存这两项最常见的需求之外，如果你还想针对其他指标进行排序，那么就没有直接的快捷键了，而是要用一种万能的方法，下面我们就来介绍这种万能的方法。

使用快捷键 F（大写字母 f）或者 O（大写字母 o），可以让我们进入到一个列选择页面，这里便可以设置我们想按照哪一列来进行排序。方法很简单，就是键入那一列前面所对应的字母，然后再按回车即可。这时，默认是按照降序来排序的，我们可以使用 R（大写字母 r）在升序和降序之间切换。

```
Current Sort Field:  K  for window 1:Def
Select sort field via field letter, type any other key to return

  a: PID       = Process Id                              Note2:
  b: PPID      = Parent Process Pid                          Field sorting
uses internal values,
  c: RUSER     = Real user name                               not those in
column display.  Thus,
  d: UID       = User Id                                    the TTY & WCHAN
fields will violate
  e: USER       = User Name                                   strict ASCII
collating sequence.
```

```
  f: GROUP     = Group Name                                    (shame on you if
WCHAN is chosen)
  g: TTY       = Controlling Tty
  h: PR        = Priority
  i: NI        = Nice value
  j: P         = Last used cpu (SMP)
* K: %CPU      = CPU usage
  l: TIME      = CPU Time
  m: TIME+     = CPU Time, hundredths
  n: %MEM      = Memory usage (RES)
  o: VIRT      = Virtual Image (kb)
  p: SWAP      = Swapped size (kb)
  q: RES       = Resident size (kb)
  r: CODE      = Code size (kb)
  s: DATA      = Data+Stack size (kb)
  t: SHR       = Shared Mem size (kb)
  u: nFLT      = Page Fault count
  v: nDRT      = Dirty Pages count
  w: S         = Process Status
  x: COMMAND   = Command name/line
  y: WCHAN     = Sleeping in Function
  z: Flags     = Task Flags
```

好了，有了万能方法，我们就可以随心所欲地对 top 中的信息进行各种排序了！

附送几个好用的快捷键吧

在 top 页面中，我们使用的快捷键如表 14 所示，可以实现一些很实用的效果，建议大家自己试用一下。

表 14　快捷键

快捷键	作　　用
b	会将当前处于运行（R）状态的进程高亮显示
d	键入 d 后，会出现提示词，再输入一个数字，则表示设置 top 页面刷新的间隔秒数。我常将这个值从默认的 3 秒调整为 1 秒，这样可以更快地看到系统的最新状态
m	注意看系统级数据部分，这个快捷键是内存和交换分区的信息展示开关。按一下就显示，再按一下就消失
l	小写的 L。和 m 类似，这个快捷键是系统负载信息的显示开关
t	和 m、l 类似，这个快捷键则是 CPU 和进程统计信息的显示开关。为了方便记忆，你可以把 l、m、t 记成“流氓兔”。这三个按钮可以控制系统级数据的显示与否
c	注意看 COMMAND 列，这个快捷键可以控制 COMMAND 列是否展示完整的命令行

不贪多，这一段落，大家能记住记这几个快捷键就已经很好了，“流氓兔”可以当作一组来记忆，这样可以提高记忆的效率。

10 top 命令庖丁解牛之五——CPU 和内存

多核时代，top 不掉队

2005 年，可以算是多核处理器的元年。当年，英特尔率先出击，推出了划时代的桌面级处理器 D820/830/840。紧随其后，AMD 也发布了 Athlon64 X2 系列双核处理器。这标志着处理器进入了多核时代。

如今，多核处理器已经成为了市场上的标配，我们的运维工程师们也需要了解多核处理器的运行状态。而 top 恰恰提供了很方便的查看方法。

在 top 页面，简单地按一下数字 1，则每一个 CPU 核心的单独的性能数据就已经赫然展现在你的眼前啦。

```
top - 16:13:00 up 77 days, 23:34,  1 user,  load average: 0.08, 0.06, 0.08
Tasks: 168 total,   1 running, 167 sleeping,   0 stopped,   0 zombie
Cpu0  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu1  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu2  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3  :  0.0%us,  0.3%sy,  0.0%ni, 99.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   3104152k total,  1769324k used,  1334828k free,   292008k buffers
Swap:        0k total,        0k used,        0k free,   353304k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
43666 roc       20   0 15028 1356  992 R  0.3  0.0   0:00.02 top
    1 root      20   0 19224 1096  812 S  0.0  0.0   0:09.11 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      RT   0     0    0    0 S  0.0  0.0   0:00.81 migration/0
    4 root      20   0     0    0    0 S  0.0  0.0   0:04.35 ksoftirqd/0
```

从所看到的 Cpu0、Cpu1、Cpu2、Cpu3 这几行，我们可以推断出这台服务器拥有 4 个 CPU 核心（假如开启了超线程技术，那么可能会包含虚拟的 CPU 核心），而且可以很清晰地看出每一个 CPU 核心当前的性能消耗情况。

当某一个 CPU 核心的负载很高时

如果你发现某一个 CPU 核心（假如是 Cpu2）的负载很高，这时候应该怎么办呢？

其实不难。

首先，我们通过快捷键 f 进入列选择页面，并将“j: P = Last used cpu (SMP)”这一列选中展示出来。

然后，回到 top 页面，并通过快捷键 b 将正在运行的进程高亮显示。

静静地观察，我们就能观察到运行在 Cpu2 上的进程了。接下来，就可以通过查看/proc 中此进程的相关信息继续深入寻找线索了。

可能的原因之一便是，这个程序的开发者使用了“CPU 核绑定函数”，将此程序或者其线程绑定到了 Cpu2 核心上，从而导致了这个问题。

至于解决方法嘛，或者 kill 掉，或者优化程序，或者解绑 CPU 核，都是不错的方法。

带你看懂进程的内存指标

在 top 命令中，我们看到了不少和内存相关的指标，它们是 VIRT/RES/SHR/DATA/CODE/SWAP/%MEM。估计很少有人真的深究过这几个指标的含义和作用，今天，我们希望通过这个段落，能让大家对这几个指标有更清晰地了解。

众所周知，Linux 系统的/proc 中有很多以数字命名的目录，这些数字对应着当前 Linux 系统中的每一个进程。而每个目录中，都囊括了对应进程的丰富的信息，其中的 statm 文件，便是我们这次要介绍的主人公，因为 top 命令输出中的 VIRT/RES/SHR 都是读取自这个文件。

首先在/proc 中随便挑选一个进程，查看它的 statm 文件，一起感受一下：

```
[roc@roclinux 35631]$ cat statm
6513 411 274 83 0 168 0
```

statm 文件的内容很简单也很朴素，没有格式的修饰，甚至连一个辅助提示信息都没有，只有 7 组光秃秃的数字。

这 7 个数字，分别代表的是：

- 6513：VIRT。
- 411：RES。
- 274：SHR。
- 83：Trs。

- 0：Lrs。
- 168：Drs。
- 0：Dt。

看，VIRT/RES/SHR 都在里面了。需要注意的是，这里显示的数字的单位都是 4KB（内存管理的最小单位：物理页面），而你通过 top 命令看到的数字的单位则是 KB，这里面存在一个四倍的转换关系。

如果你对 Linux 内核有一定研究，那么你将有能力追溯出 VIRT/RES/SHR 的计算方法和来源，并且能深刻理解它们的含义和作用。

而对于不太了解 Linux 内核的同学们，我们就不卖关子啦，直接和大家介绍这几个关键的概念：

- VIRT，英文全称是 Virtal Memory，中文为虚拟内存。一个进程，无论是通过 malloc/calloc 系列函数申请的内存，还是堆/栈所占用的内存，抑或是全局变量等所占用的内存，都属于 VIRT 范畴。所以 VIRT 是进程所占用内存的最大集。
- RES，英文全称是 Resident Memory，中文叫作常驻内存。因为大家知道，在 malloc 申请了物理内存空间之后，并非会立即使用到这块物理内存，所以，系统会在进程真正要使用到这块物理内存时，才正式将物理内存分配给这个进程。所以，RES 表示的是一个进程真正在使用的物理内存的大小，而非申请的物理内存的大小。
- SHR，英文全称是 Shared Memory，中文叫作共享内存。这个指标，表示这个进程中有多少内存是被共享的，准确地说是“可能”被共享的。
- CODE，表示进程的程序的机器指令部分的大小。这部分也叫作 Trs，即 Text Resident Set。
- DATA，其实对应的是除了 CODE 之外的内存空间，也叫作 Drs，即 Data Resident Set。
- SWAP，进程被置换的虚拟内存空间大小。

而%MEM 的含义，用一个公式或许更容易让大家理解，那就是

```
%MEM = ( 常驻内存 / 总内存大小 ) × 100%
```

至此，我们的 top 命令讲解就结束了，我们一起学习了 top 的列管理、系统数据、进程数据、排序大法，还有 CPU 和内存的深度知识，我相信，这足以让各位同学们去生产环境中一试身手，去尝试追查一些实际的线上故障啦，勇敢地出发吧！

11 iostat 让 I/O 尽在掌握之中

认识 iostat

iostat，用来显示 CPU 的统计信息以及整个系统、适配器、tty 设备和硬盘的输入/输出信息。

首先，我们来看看 iostat 执行起来的样子：

```
[root@roclinux ~]# iostat
Linux 2.6.32-431.el6.x86_64 (root)   03/02/2016   _x86_64_(2 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          0.36    0.00    0.35    0.00    0.01   99.28

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
vda              0.11         0.21         4.92     454178   10585768
vdb              0.00         0.01         0.00      16008          0
```

可以看到，iostat 的输出信息主要分为三大部分。

- 第一部分：系统信息。

```
Linux 2.6.32-431.el6.x86_64 (roc)    03/02/2016   _x86_64_(2 CPU)
```

包括了系统名称、内核版本信息以及架构信息等。

- 第二部分：CPU 信息。

```
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          0.36    0.00    0.35    0.00    0.01   99.28
```

阅读过前面 top 系列文章的同学，相信对这几个指标项并不陌生。

- 第三部分：磁盘信息。

```
Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
vda              0.11         0.21         4.92     454178   10585768
vdb              0.00         0.01         0.00      16008          0
```

这里展示出了本台服务器上各个存储设备的 I/O 情况，包括块的读写量和读写速度等。

能看 TPS 和吞吐量吗？

当我们怀疑磁盘出现性能问题时，通常需要查看磁盘的各种信息。磁盘的工作状态如何？TPS 是多少？吞吐量有没有问题？我们需要通过命令把这些信息展示出来：

```
[root@roclinux ~]# iostat -d -k 1 3
Linux 2.6.32-431.el6.x86_64 (root)   01/23/2016   _x86_64_(2 CPU)

Device:            tps   kB_read/s   kB_wrtn/s   kB_read   kB_wrtn
vda              0.74        8.37        6.21    135937    100872
vdb              0.13        0.52        0.00      8472         0

Device:            tps   kB_read/s   kB_wrtn/s   kB_read   kB_wrtn
vda              0.00        0.00        0.00         0         0
vdb              0.00        0.00        0.00         0         0

Device:            tps   kB_read/s   kB_wrtn/s   kB_read   kB_wrtn
vda              0.00        0.00        0.00         0         0
vdb              0.00        0.00        0.00         0         0
```

上面的命令显示了磁盘读写数据的情况，命令中的选项和参数有必要和大家解释一下：

- -d 选项：表示只显示磁盘的使用状态，指定该选项后，iostat 将只显示磁盘信息而不再显示 CPU 信息。
- -k 选项：表示使用统 KB 作为单位，比如 kB_read/s 列的 8.37 表示的就是 8.37KB/s。
- “1”参数：表示采样时间，本例为 1 秒。
- “3”参数：表示采样次数，本例为 3 次。

那上面显示的指标都是什么呢？有点懵。没有关系，我们一起来看看如表 15 所示的详细解释。

表 15　指标解释

名　称	解　释
tps	每秒进程的 I/O 读、写请求总数
kB_read/s	每秒读取的字节数（单位为 KB）
kB_wrtn/s	每秒写入的字节数（单位为 KB）
kB_read	读取的字节总数（单位为 KB）
KB_wrtn	写入的字节总数（单位为 KB）

有了这些指标，你就可以轻轻松松地了解磁盘的当前性能了。不过，细心的同学

会发现，这个示例里的第一组数据显示的 I/O 消耗很大，而下面的两组数据却都为 0，难道真的是恰好 I/O 波动这么大么？

可疑的第一组数据

iostat 输出中的第一组数据，是有特殊意义的，它表示从 Linux 系统启动到本命令执行这段时间的统计结果，而后面的数据都表示 iostat 执行一个采样周期（通常是 1 秒）的统计结果。这样看来，第一组数据外的其他数据才是实时的、有效的、能反映最近磁盘 I/O 情况的。这也是为什么上一段落的示例中后两组的数据都是 0 而第一组数据那么高的原因了。

```
#为了排除第一组数据的干扰，可以使用-y选项，它可以实现不显示iostat的第一组数据：
[root@roclinux ~]# iostat -d -k -y 1 3
Linux 2.6.32-431.el6.x86_64 (root) 03/02/2016  _x86_64_    (2 CPU)

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
vda              0.00         0.00         0.00          0          0
vdb              0.00         0.00         0.00          0          0

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
vda              0.00         0.00         0.00          0          0
vdb              0.00         0.00         0.00          0          0

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
vda              0.00         0.00         0.00          0          0
vdb              0.00         0.00         0.00          0          0
```

你了解 util 和 await 么

前面我们只是查看了磁盘的读写速度，系统变慢的原因还是没有找到，我们还需要更多的指标线索：

```
[root@roclinux ~]# iostat -d -x -k 1 3
Linux 2.6.32-431.el6.x86_64 (root)   01/23/2016   _x86_64_(2 CPU)

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz
avgqu-sz   await  svctm  %util
vda               0.09     0.80    0.31    0.14     4.69     3.76    37.38
0.00    2.77   0.31   0.01
vdb               0.00     0.00    0.07    0.00     0.29     0.00     8.10
0.00    0.83   0.82   0.01

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz
avgqu-sz   await  svctm  %util
vda               0.00     0.00    0.00    0.00     0.00     0.00     0.00
0.00    0.00   0.00   0.00
```

```
vdb               0.00     0.00    0.00    0.00     0.00     0.00     0.00
0.00    0.00   0.00   0.00

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz
avgqu-sz   await  svctm  %util
vda               0.00     0.00    0.00    0.00     0.00     0.00     0.00
0.00    0.00   0.00   0.00
vdb               0.00     0.00    0.00    0.00     0.00     0.00     0.00
0.00    0.00   0.00   0.00
```

-x 选项展示出了更多的硬盘统计数据，那么，这些指标都代表什么含义呢？如表 16 所示。

表 16　指标的含义

name	comment
rrqm/s	每秒对该设备的读请求被合并次数，文件系统会对读取同块（block）的请求进行合并
wrqm/s	每秒对该设备的写请求被合并次数，文件系统会对写入同块（block）的请求进行合并
r/s	每秒完成读 I/O 的次数
w/s	每秒完成写 I/O 的次数
rsec/s	每秒读扇区数，每扇区 512 字节
wsec/s	每秒写扇区数，每扇区 512 字节
rkB/s	每秒读千字节数
wkB/s	每秒写千字节数
avgrq-sz	平均每次 I/O 操作的数据大小（扇区），即(rsec/s+wsec/s)/(r/s+w/s)
avgqu-sz	平均等待处理的 I/O 请求队列长度
await	平均每次 I/O 请求等待时间（包括等待时间和处理时间，毫秒为单位），可以理解为 I/O 的响应时间。一般系统的 I/O 响应时间应该低于 5ms，如果大于 10ms 就比较大了
svctm	平均每次 I/O 操作的服务时间，单位毫秒
%util	周期内用于 I/O 操作的时间比率，即 I/O 队列非空的时间比率，即(r/s+w/s)*(svctm/1000)

了解这些指标的含义后，我们还是看不出这些数字中隐藏的线索和问题啊？没有关系，下面就把我们在实际排查问题时的一些经验总结奉献给大家：

- 如果%util 较大，则代表 I/O 请求太多，硬盘可能存在瓶颈。
- await 大于 svctm，差值越小，则说明队列时间越短；反之差值越大，则队列时间越长，说明系统出了问题。
- 如果 svctm 比较接近 await，则说明 I/O 几乎没等待时间。

- 如果 await 远大于 svctm，则说明 I/O 队列太长，应用响应时间也变长。
- avgqu-sz 队列长度也可衡量 I/O 负荷的指标，avgqu-sz 是单位时间内的平均值。

最后再来看看 CPU

除了看磁盘外，CPU 也可能是导致系统变慢的原因，现在我们就去看看 CPU。

```
[root@roclinux ~]# iostat -c 1 3
Linux 2.6.32-431.el6.x86_64 (root)   01/23/2016   _x86_64_(2 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          0.08    0.00    0.09    0.01    0.01   99.81

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          0.00    0.00    0.00    0.00    0.00  100.00

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          0.00    0.00    0.00    0.00    0.00  100.00
```

和-d 选项类似，-c 选项表示显示 CPU 使用状态，输出的 CPU 相关指标含义也列在表 17 中了

表 17 CPU 相关指标含义

name	comment
%user	显示了 CPU 在用户级（应用程序）执行时产生的时间百分比
%nice	显示了 CPU 在应用程序使用 NICE 权限执行时产生的时间百分比
%system	显示了 CPU 在系统级（内核）执行时产生的时间百分比
%iowait	显示了 CPU 空闲且系统有未完成的磁盘 I/O 请求的时间百分比
%steal	显示了 hypervisor 服务另一个虚拟处理器时，虚拟 CPU 等待实际 CPU 的时间百分比
%idle	显示了 CPU 空闲且系统没有未完成的磁盘 I/O 请求的时间百分比

注意：

1. NICE 表示进程可被执行的优先级的修正数值。它影响进程的优先级：PRI(new)=PRI(old)+nice。这样，当 nice 值为负值时，该程序优先级值将变小，优先级会变高而得以执行

2. 如果 Steal 值比较高的话，你就需要向主机供应商申请扩容虚拟机。这是因为服务器上的另一个虚拟机可能拥有更大更多的 CPU 时间片，从而占用了你的虚拟机的 CPU 的时间。

3. 如果升级了虚拟机后，steal 值还是不下降的话，请尽快逃离，这说明该主机供应商过量出售它的虚拟机。

最后，还是给出实际运维过程中的一些经验总结：

- %iowait 的值过高，表示硬盘存在 I/O 瓶颈。
- %idle 值高，表示 CPU 较空闲。
- %idle 值高但系统响应慢时，有可能是 CPU 等待分配内存，此时应加大内存容量。
- %idle 小于 70，I/O 压力就较大了，一般读取速度有较多的 wait。
- %idle 值如果持续低于 10，那么系统的 CPU 处理能力相对较低，表明系统中最需要解决的资源是 CPU。

注意，本文中所涉及的经验总结仅供参考，不同的运维环境情况千差万别，还请具体情况具体分析，关键是一定要透彻掌握各个指标的含义。

12 让 pidof 告诉我们进程 ID

如何知道程序的进程 ID

就像我们每个人都拥有一个唯一的身份证号一样，Linux 系统中每一个运行的程序也都拥有一个唯一的进程 ID，简称 PID，也就是 Process ID。我们很多的操作都需要依据 PID 来实现，那么我们如何才能知道一个运行的程序的 PID 呢，本文就来说说这个话题。

如果你知道 pidof 命令的话，那么找到 PID 就并非难事了。比如我们想知道一台 Linux 服务器上运行的 sshd 程序的 PID，那么可以用下面的命令来实现。

```
[root@roclinux ~]# pidof sshd
21559 21555 1319
```

可以看出，pidof 列出了三个 PID，说明这台 Linux 服务器上有 3 个进程都是由 sshd 程序启动的。

找出 Shell 脚本的 PID

如果你阅读完第一个段落就浅尝辄止，那么很可惜，你可能永远也不会知道“pidof 如何才能找到 Shell 程序的 PID”的方法了。

对于坚持打破砂锅问到底的同学，我们为你奉上这个知识点：

```
#我们编写了一个Shell脚本
[root@roclinux ~]# cat while.sh
#! /bin/bash
sleep 1m

#启动脚本，并置于后台运行
[root@roclinux ~]# ./while.sh &
[1] 13902

#直接用pidof是查找不到的
[roc@roclinux 20160410]$ pidof while.sh
[roc@roclinux 20160410]$
```

```
#一定要用-x选项，才可以找到Shell脚本的PID
[root@roclinux ~]# pidof -x while.sh
13902
```

只显示程序的一个进程 ID

在前面的第一个示例中，显示了三个由 sshd 程序启动的 PID，如果我们只希望显示出一个，那么就用-s 选项。

```
# -s选项，只会输出其中一个PID
[root@roclinux ~]# pidof -s sshd
21559
```

忽略无用的 PID

如果觉得某些 PID 对我们是干扰，那么可以让 pidof 命令不展示特定的 PID：

```
# 使用-o选项，则会忽略我们指定的PID
[root@roclinux ~]# pidof sshd -o 21559
21555 1319
```

我们用-o 选项，告诉 pidof 不要展示 21559 这个 PID，只展示其他 PID 就好了。

pidof 和 killall5 的关系

如果我告诉你 pidof 和 killall5 两个命令是一母同胞，你会相信么？

什么？这怎么可能？这两个命令从功能上可是风马牛不相及呀！

好，现在就证明给大家看。

```
[root@roclinux ~]# ls -l /sbin/pidof
lrwxrwxrwx. 1 root root 8 Sep  1  2014 /sbin/pidof -> killall5
[root@roclinux ~]# ls -l /sbin/killall5
-rwxr-xr-x. 1 root root 15184 Jun 29  2013 /sbin/killall5
```

从上面的命令输出可以看出，pidof 和 killall5 竟然是同一个可执行程序。

如此功能迥异的两个命令，竟然复用的是同一个程序，这究竟是为什么呢？

通过深入阅读 killall5 的源代码，我们知道了答案。原来程序中有一个叫作 readproc() 的函数，它实现了遍历/proc 目录并从中寻找匹配进程的功能，而 pidof 和 killall5 都用到了这个函数所提供的功能，因此，pidof 和 killall5 复用了这一程序。

说说 pidof 的返回值

pidof 命令的返回值，一般有两个：

- 0：表示 pidof 至少找到了一个对应的 PID。
- 1：表示 pidof 没有找到任何对应的 PID。

```
#找一找vim程序对应的PID
[root@roclinux ~]# pidof vim
11700 11221

#返回值为0，表示匹配到至少一个PID
[root@roclinux ~]# echo $?
0

#找一找ssd程序对应的PID
[root@roclinux ~]# pidof ssd

#返回值为1，表示没有找到
[root@roclinux ~]# echo $?
1
```

通过返回值我们可以变相地判断某个进程是否还在运行。

好了，pidof 是一个精致小巧的命令，很实用也不难学，相信学完本节后再也不用为寻找 PID 而发愁了。

13 sar 访谈

我们今天的嘉宾是 sar

今天为大家介绍一位新朋友，它的名字叫作 sar，它的脑子里装满了服务器系统性能的信息。你只要和它搞好关系，以后想获得任何这方面的信息，就都不用发愁了！

sar 命令有两个参数非常非常实用，那就是“时间间隔”和“输出次数”：

- 时间间隔：表示两次信息输出之间的时间间隔，单位是秒。如果这个值被设置为 0，则表示所输出的信息是从开机到现在为止的信息平均值。如果大于 0，那么 sar 命令所输出的便是从当前开始计算的一段时间内（你所设置的间隔时间）的平均值。
- 输出次数：表示输出信息的次数，默认情况下是 1 次。如果这个值被设置为 0，则 sar 会永远地输出下去，直到你按下组合键 Ctrl+C 为止。

我们来看一下 sar 命令运行的样子：

```
[roc@roclinux ~]$ sar 2 3
Linux 2.6.32-220.4.1.el6.x86_64 (roclinux)       2016年03月30日   _x86_64_
(4 CPU)

17时36分58秒     CPU    %user    %nice   %system   %iowait    %steal    %idle
17时37分00秒     all     3.19     0.00      0.38      0.00      0.00     96.42
17时37分02秒     all     3.52     0.00      0.65      0.00      0.00     95.83
17时37分04秒     all     0.00     0.00      0.25      0.00      0.00     99.75
平均时间:        all     2.21     0.00      0.43      0.00      0.00     97.36
[roc@roclinux ~]$
```

其中，sar 2 3 表示每 2 秒输出一次信息，共输出 3 次实时信息，再外加一行汇总的平均值，总共是 4 行信息。

而输出的内容都是和 CPU 相关的性能指标，包括我们熟悉的 user、nice、system、iowait、steal 和 idle。

如果你对这些指标项还不是很了解的话，建议你先阅读一下 vmstat 命令和 top 命令等相关文章。

sar 的输出可以存到文件中么

为了让运维人员可以存储现场数据，以备事后追查和存档，sar 命令很体贴地提供了一个-o 选项。只要使用它，就可以把输出内容保存到文件中了。

下面来看一个示例。

```
#我们让sar把输出保存到op_info文件中
[roc@roclinux 20160330]$ sar 2 3 -o op_info
Linux 2.6.32-220.4.1.el6.x86_64 (roclinux)     2016年03月30日  _x86_64_
(4 CPU)

17时44分56秒
CPU    %user    %nice  %system  %iowait   %steal    %idle
17时44分58秒    all     0.00     0.00     0.27     0.14     0.00    99.59
17时45分00秒    all     0.00     0.00     0.12     0.00     0.00    99.88
17时45分02秒    all     0.00     0.00     0.26     0.00     0.00    99.74
平均时间:       all     0.00     0.00     0.21     0.04     0.00    99.74

#好消息，文件已经生成啦!
[roc@roclinux 20160330]$ ls -lh op_info
-rw-r--r-- 1 roc roc 71K 3月  30 17:45 op_info

#可是当我们想查看它的时候，却发现……
[roc@roclinux 20160330]$ cat op_info
??p    ??VLinuxroclinux2.6.32-220.4.1.el6.x86_64x86_64? ??P@
                                                          P
堝@@p` ( PX??V,8句loE嫜                                    H

  枧贵溉K?=??????8?"躐 1蹁????]??4钪-??Di>た?%氓m'喂????kO饺X]
```

意外发生了，当我们用 cat 来查看 op_info 文件时发现展示出来的全都是乱码，这可如何是好？

别着急，这里涉及了一个小知识，sar 命令在存储输出内容时采用了结构化的方式，并非纯文本格式，所以在读取这类文件时，不能直接用 cat 命令读取。为此，sar 特意为我们准备了-f 选项，你只要用-f 选项指定要展示的信息存储文件，就可以清晰地展示出之前保存的输出信息了。

```
[roc@roclinux ~]$ sar -f op_info
Linux 2.6.32-220.4.1.el6.x86_64 (roclinux)     2016年03月30日  _x86_64_
(4 CPU)
```

```
17时44分56秒     CPU    %user     %nice   %system   %iowait    %steal     %idle
17时44分58秒     all     0.00      0.00      0.27      0.14      0.00     99.59
17时45分00秒     all     0.00      0.00      0.12      0.00      0.00     99.88
17时45分02秒     all     0.00      0.00      0.26      0.00      0.00     99.74
平均时间:        all     0.00      0.00      0.21      0.04      0.00     99.74
[roc@roclinux ~]$
```

有一个粗心的小伙伴，他使用 root 账户执行 sar 命令，在使用-o 选项时，竟然忘记指定文件名了，就像这样：

```
[root@roclinux ~]# sar 2 3 -o
Linux 2.6.32-220.4.1.el6.x86_64 (roclinux)       2016年03月30日   _x86_64_
(4 CPU)

17时54分51秒     CPU    %user     %nice   %system   %iowait    %steal     %idle
17时54分53秒     all     0.00      0.00      0.13      0.13      0.00     99.75
17时54分55秒     all     0.00      0.00      0.12      0.12      0.00     99.75
17时54分57秒     all     0.00      0.00      0.00      0.00      0.00    100.00
平均时间:        all     0.00      0.00      0.08      0.08      0.00     99.83
```

好像和不加-o 选项效果一样，似乎什么都没有发生。但其实并非如此，sar 命令规定，在-o 选项后没有指定文件时，sar 会自动把输出信息存储到/var/log/sa/saDD 文件里，而其中的 DD 则表示当天的日期数字。今天是 30 日，我们去看看是不是真的有/var/log/sa/sa30 文件呢？

```
#果然有这个文件
[root@roclinux ~]# ls -l /var/log/sa/sa30
-rw-r--r-- 1 root root 268196 3月  30 17:58 /var/log/sa/sa30

#我们来看看它的内容
[root@roclinux sa]# sar -f sa30
Linux 2.6.32-220.4.1.el6.x86_64 (roclinux)       2016年03月30日   _x86_64_
(4 CPU)

18时00分48秒     CPU    %user     %nice   %system   %iowait    %steal     %idle
18时00分50秒     all     0.00      0.00      0.27      0.94      0.00     98.79
18时00分52秒     all     0.00      0.00      0.26      0.13      0.00     99.62
18时00分54秒     all     0.00      0.00      0.37      0.00      0.00     99.63
平均时间:        all     0.00      0.00      0.30      0.34      0.00     99.36
```

可以看到，输出信息真的存储到了/var/log/sa/sa30 文件里。需要注意的一点是，-o 选项采用的是追加的方式来写入文件，并不会删除文件中之前的 sar 记录。

另外，在使用-o 指定文件时，sar 命令也是有要求有底线的，如果你指定的文件本身既不是一个全新的文件，又不是 sar 格式的文件，那么 sar 的输出是这样的。

```
#一个纯文本文件
[roc@roclinux 20160330]$ cat op_info
hello

#我们指定这个纯文本文件作为存储文件
[roc@roclinux 20160330]$ sar 2 3 -o op_info
无效的系统运行记录文件: op_info
无效的数据格式
```

看到了吧，sar 早就发现了蹊跷之处，它会停止运行，不向里面写入任何内容。

sar 命令支持多核 CPU 么

哈哈，完全没问题的，sar 命令拥有一个-P 选项（大写字母 P），就是专门用来展示多核处理器性能指标的。

当使用 sar 命令而没有设定-P 选项时，sar 会根据所有的 CPU 核信息给出一个汇总报告。

当使用-P ALL 时，sar 命令就会针对每一个 CPU 核都给出其具体性能信息，然后再给出一个总的性能信息。

比如，我这里有一个至强处理器的 CPU，是八核的，看看-P 选项的威力吧。

```
#我们让sar命令展示每一个CPU核的信息，以1秒为间隔，总共展示1次
[roc@roclinux 20160330] sar -P ALL 1 1
Linux 2.6.9   03/30/2016

10:59:38 PM      CPU    %user    %nice   %system   %iowait    %idle
10:59:39 PM      all     2.12     0.00     2.87     0.00     95.01
10:59:39 PM        0     0.00     0.00     1.98     0.00     98.02
10:59:39 PM        1     9.00     0.00     7.00     0.00     84.00
10:59:39 PM        2     0.00     0.00     1.98     0.00     98.02
10:59:39 PM        3     7.00     0.00     1.00     0.00     92.00
10:59:39 PM        4     0.00     0.00     3.03     0.00     96.97
10:59:39 PM        5     0.00     0.00     1.00     0.00     99.00
10:59:39 PM        6     0.00     0.00     3.96     0.00     96.04
10:59:39 PM        7     0.99     0.00     1.98     0.00     97.03

Average:         CPU    %user    %nice   %system   %iowait    %idle
Average:         all     2.12     0.00     2.87     0.00     95.01
Average:           0     0.00     0.00     1.98     0.00     98.02
Average:           1     9.00     0.00     7.00     0.00     84.00
Average:           2     0.00     0.00     1.98     0.00     98.02
Average:           3     7.00     0.00     1.00     0.00     92.00
Average:           4     0.00     0.00     3.03     0.00     96.97
Average:           5     0.00     0.00     1.00     0.00     99.00
Average:           6     0.00     0.00     3.96     0.00     96.04
Average:           7     0.99     0.00     1.98     0.00     97.03
```

看到了吧，sar 命令把每一个 CPU 核都编上了号（从 0 到 7），并且依次展示了它们的各个性能指标。

不过这样展示出来的信息太多，不太便于阅读。其实 sar 命令也想到了这个问题，所以还支持使用-P 选项来明确指定某一个 CPU 核，然后只针对这个单独的 CPU 核输出具体的性能信息。当我们想查看第 6 个 CPU 核的信息时，可以这样来做。

```
#因为CPU核是从0开始编号的，所以第6个CPU核的编号就是5
[roc@roclinux 20160330] sar -P 5 1 1
Linux 2.6.9     03/30/2016

11:01:16 PM       CPU     %user     %nice   %system   %iowait     %idle
11:01:17 PM         5      0.00      0.00      2.00      0.00     98.00

Average:          CPU     %user     %nice   %system   %iowait     %idle
Average:            5      0.00      0.00      2.00      0.00     98.00
```

sar 还可以展示哪些性能信息

sar 命令在默认情况下是输出 CPU 信息的，但可不要小瞧了 sar 命令，在查看内存、网络、I/O 等方面，sar 也是我们最好的得力助手。下面让我们来领略一下它的各项本领。

我们可以使用这些选项来让 sar 展示更丰富的信息：

- -b 选项：报告 I/O 使用情况以及传输速率。（只适用于 2.5 及之前的内核，所以新内核有可能不支持这个选项）
- -B 选项：报告“页”使用情况。
- -c 选项：报告进程创建情况。
- -d 选项：报告每一个块设备的使用情况（当使用时，你会发现在 DEV 列有类似 dev1-7 格式的字符串，其中 1 代表设备的主序号，n 代表设备的从序号，而且 rd_sec/s 列和 wr_sec/s 列的单位都是 512bytes，即 512B，也就是 0.5KB）
- -I 选项：汇报中断情况。
- -n 选项：汇报网络情况。
- -P 选项：设定 CPU。
- -q 选项：汇报队列长度和负载信息。
- -r 选项：汇报内存和交换区使用情况。
- -R 选项：汇报内存情况。
- -u 选项：汇报 CPU 使用情况。

- -v 选项：汇报 i 节点、文件和其他内核表信息。
- -w 选项：汇报系统上下文切换情况
- -x 选项：可以针对某个特定 PID 给出统计信息，可以直接指定进程 ID 号，也可以指定为 SELF，这样就是检测 sar 进程本身。如果设定为 ALL，则表示汇报所有系统进程信息。
- -X 选项：汇报特定 PID 的子进程的信息。
- -y 选项：设定 TTY 设备的信息。

看了这一段后，你是不是对 sar 命令肃然起敬了，如果要为 sar 命令设计一句广告词的话，“服务器性能数据，360 度查看无死角”最合适不过啦。

虽然 sar 命令这么强大，但如此多的选项提高了 sar 命令的学习门槛，大部分同学是记不住这么多选项的。没关系，sar 为我们提供了一个超级无敌的选项，那就是 -A 选项，它可以用来展示所有 sar 所知道的服务器性能信息。

```
[roc@roclinux 20160330]$ sar -A
Linux 2.6.32-220.4.1.el6.x86_64 (roclinux)      2016年03月30日   _x86_64_
(4 CPU)

18时00分48秒
CPU     %usr    %nice     %sys   %iowait    %steal     %irq    %soft    %guest
    %idle
18时00分50秒     all      0.00      0.00      0.27      0.94      0.00      0.00
0.00     0.00    98.79
18时00分50秒       0      0.00      0.00      0.00      3.19      0.00      0.00
0.00     0.00    96.81
18时00分50秒       1      0.00      0.00      0.00      0.00      0.00      0.00
0.00     0.00   100.00
(此处省略上千行...)
```

-A 选项其实就相当于-bBcdqrRuvwWy -I SUM -n FULL -P ALL 这么一堆选项的集合，擦亮眼睛看看每一个选项，几乎涵盖了你想了解的绝大多数信息。

不过，话说回来，-A 选项轻易还是不要用为好，如此海量的信息输出到屏幕上，刷屏就要刷好久，实在是不适合人类阅读。

为 sar 指定结束时间

为 sar 指定间隔时间和输出次数，是最常用的方法，但是在有些场景下，我们并不希望这样做，而是想让 sar 在某个特定时间结束，以便可以记录一个时间段内的性能变化情况。

这时，我们使用-e 选项来指定 hh:mm:ss 时间点就可以了，设定好你的结束时间，sar 就会乖乖地在这个时间结束喽。

默认情况下结束时间是 18:00:00，也就是说，如果你使用-e 但没有指定具体时间，那么 sar 就会在下午 6 点结束。

需要注意的一点是，-e 选项常常搭配-o 选项或-f 选项来使用：

- 搭配-o 选项：可以用来指定存储结束的时间点。
- 搭配-f 选项：可以用来指定从文件读取的数据的结束时间点。

```
[roc@roclinux ~]$ sar -e 19:30:00 -o record.log 2
Linux 2.6.32-220.4.1.el6.x86_64 (roclinux)     2016年03月30日  _x86_64_      (4
CPU)

18时31分13秒    CPU    %user    %nice   %system   %iowait    %steal    %idle
18时31分15秒    all    10.38     0.00     0.62     0.00     0.00    89.00
18时31分17秒    all     0.14     0.00     0.14     0.00     0.00    99.72
```

用 sar 来查看网络信息

sar 命令使用-n 选项汇报网络相关信息，可用的参数包括：

- DEV
- EDEV
- SOCK
- FULL

如果使用 DEV 关键字，那么 sar 将汇报和网络设备相关的信息，如 lo、eth0 或 eth1 等，例如：

```
#查看所有网络接口的性能信息
[roc@roclinux ~]$ sar -n DEV 1 2
Linux 2.6.9     03/30/2016

12:10:49 AM     IFACE   rxpck/s   txpck/s   rxbyt/s   txbyt/s   rxcmp/s
txcmp/s  rxmcst/s
12:10:50 AM      eth0     63.64      0.00   4072.73      0.00      0.00
0.00     0.00
12:10:50 AM      eth1     30.30     13.13   2907.07   1234.34      0.00
0.00     0.00
12:10:50 AM        lo      0.00      0.00      0.00      0.00      0.00
0.00     0.00
```

其中的指标说明如下：

- IFACE：就是网络设备的名称。
- rxpck/s：每秒钟接收到的包数目。
- txpck/s：每秒钟发送出去的包数目。
- rxbyt/s：每秒钟接收到的字节数。
- txbyt/s：每秒钟发送出去的字节数。
- rxcmp/s：每秒钟接收到的压缩包数目。
- txcmp/s：每秒钟发送出去的压缩包数目。
- txmcst/s：每秒钟接收到的组播包的包数目。

如果使用 EDEV 关键字，那么 sar 命令会针对网络设备汇报其失败情况，例如：

```
[roc@roclinux 20160330]$ sar -n EDEV 1 3
Linux 2.6.9    03/30/2016

12:15:06 AM     IFACE   rxerr/s   txerr/s    coll/s  rxdrop/s  txdrop/s
txcarr/s  rxfram/s  rxfifo/s  txfifo/s
12:15:07 AM        lo      0.00      0.00      0.00      0.00      0.00
0.00     0.00      0.00      0.00
12:15:07 AM      eth0      0.00      0.00      0.00      0.00      0.00
0.00     0.00      0.00      0.00
12:15:07 AM      eth1      0.00      0.00      0.00      0.00      0.00
0.00     0.00      0.00      0.00
```

- rxerr/s：每秒钟接收到的损坏的包的数目。
- txerr/s：当发送包时，每秒钟发生的错误数。
- coll/s：当发送包时，每秒钟发生的冲撞（collisions）数（这个只有在半双工模式下才有）。
- rxdrop/s：由于缓冲区满，网络设备接收端每秒钟丢掉的网络包的数目。
- txdrop/s：由于缓冲区满，网络设备发送端每秒钟丢掉的网络包的数目。
- txcarr/s：当发送数据包时，每秒钟载波错误发生的次数。
- rxfram/s：在接收数据包时，每秒钟发生的帧对齐错误的次数。
- rxfifo/s：在接收数据包时，每秒钟缓冲区溢出错误发生的次数。
- txfifo/s：在发送数据包时，每秒钟缓冲区溢出错误发生的次数。

如果你使用 SOCK 关键字，则会针对 socket 连接进行汇报，例如：

```
[roc@roclinux 20160330]$ sar -n SOCK 1 3
Linux 2.6.9      03/30/2016

12:27:29 AM   totsck    tcpsck    udpsck    rawsck   ip-frag
12:27:30 AM       90        41         4         0         0
12:27:31 AM       90        41         4         0         0
12:27:32 AM       90        41         4         0         0
Average:          90        41         4         0         0
```

- totsck：被使用的 socket 的总数目。
- tcpsck：当前正在被使用于 TCP 的 socket 数目。
- udpsck：当前正在被使用于 UDP 的 socket 数目。
- rawsck：当前正在被使用于 RAW 的 socket 数目。
- ip-frag：当前的 IP 分片的数目。

如果你使用 FULL 关键字，相当于上述 DEV、EDEV 和 SOCK 三者的综合信息汇报。

好了，我们把 sar 命令在 CPU、网络等方面的内容作了比较深入地介绍，同时还展示了 sar 命令的各种能力，相信你们已经入门成功了。学好 sar 命令，并非易事，有志于此的读者，还请继续加油！

14 帮你找到幕后黑手——lsof 应用篇

引言

在“万物皆文件”的 Linux 系统中，一个能随时用来监测文件各种特征的命令是必要的，也是必须的，就像我们伟大祖国的“天眼计划”，它帮助“超能”的警察叔叔破获了多少悬疑之案啊！

于是，我遍查 Linux 的各种书籍，功夫不负有心人，终于让我把它们找了出来，它们就是本系列文章的两位主角——lsof 和 fuser。

初识 lsof

lsof，即 list open files，可以用来查看进程打开的文件、目录和套接字等一系列信息。

lsof 是 Linux 系统管理员经常使用的工具之一，建议大家使用 root 账户运行该命令，不然的话，可能会有不少信息无法显示出来。

下面我们就一起来看看 lsof 有哪些通天的本领吧！

文件名定位进程

lsof 可以通过文件名来定位打开该文件的进程，方法是执行 lsof filename，示例如下：

第一步：使用 vim 打开 test.sh 文件。

```
[root@roclinux ~]# vim test.sh
```

第二步：使用 Ctrl+Z 组合键让程序转入后台。

```
[root@roclinux ~]# vim test.sh
[1]+  Stopped                 vim test.sh
```

第三步：使用 lsof 来尝试定位这个躲到后台的程序。

```
[root@roclinux ~]# lsof test.sh
```

咦，程序怎么没有找到打开文件的进程呢？难道是书上说错了？还是 lsof 程序出现 bug 了？先别急，我们排查一下问题：

```
[root@roclinux ~]# lsof | grep test.sh
vim       11700      root    4u      REG              253,1     4096
21250 /root/.test.sh.swp
```

真相大白，原来是 vim 程序的问题，vim 会偷偷地创建一个后缀为.swp 的临时文件.test.sh.swp，以便用户临时编辑时暂存编辑内容。怪不得 lsof 找不到呢？

```
[root@roclinux ~]# lsof .test.sh.swp
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF  NODE NAME
vim     11700 root    4u   REG  253,1     4096 21250 .test.sh.swp
```

我们更换了文件，这一次 lsof 准确找到了目标进程。看来 lsof 确确实实能定位打开文件的进程。

文件描述符定位进程

既然可以通过文件名定位进程，那么同理，文件描述符也应该是可以的才对。比如我们想找出所有打开标准错误输出的进程：

```
[root@roclinux ~]# lsof -d 3
COMMAND    PID     USER   FD   TYPE           DEVICE SIZE/OFF      NODE
NAME
init         1     root    3r  FIFO              0,8      0t0      6493
pipe
udevd      371     root    3u   REG              0,5     1405      7789
/dev/.udev/queue.bin
auditd     928     root    3u  sock              0,6      0t0      8318
can't identify protocol
hsa.upgra   947     root    3w   REG            253,1        0    263336
/opt/hss/daemon.stdout.log
hsa        949     root    3w   REG            253,1        0    263336
/opt/hss/daemon.stdout.log
rsyslogd   971     root    3r   REG              0,3        0 4026532040
/proc/kmsg
dbus-daem  1097       dbus     3u   unix 0xffff880119ea0380       0t0
8725 /var/run/dbus/system_bus_socket
NetworkMa  1107       root     3u   unix 0xffff880119edb680       0t0
8759 socket
modem-man  1112       root     3u   unix 0xffff88011ce7a080       0t0
8795 socket
acpid     1123     root    3r   REG              0,3        0 4026531987
/proc/acpi/event
hald      1134 haldaemon    3r  FIFO              0,8      0t0      8882
pipe
hald-runn  1135       root     3r   unix 0xffff88011ce536c0       0t0
```

```
8916 socket
dhclient    1139        root     3u   unix  0xffff88011a944cc0       0t0
8945 socket
wpa_suppl  1142     root    3w   REG            253,1       0    131036
/var/log/wpa_supplicant.log
hald-addo   1173        root     3u   unix  0xffff88011a7bf0c0       0t0
9069 socket
hald-addo   1181  haldaemon     3u   unix  0xffff88011a9440c0       0t0
9150 socket
sshd      1296     root    3u  IPv4             9676     0t0       TCP
*:ssh (LISTEN)
ntpd        1318         ntp     3u   unix  0xffff880119547080       0t0
9741 socket
abrtd     1340     root    3r  FIFO              0,8     0t0      9825
pipe
abrt-dump  1348     root    3r   DIR             0,10       0         1
inotify
crond     1356     root    3u   REG            253,1       5    135179
/var/run/crond.pid
atd       1367     root    3uW  REG            253,1       5    135507
/var/run/atd.pid
tmux      10934     root    3u   REG              0,9       0      3786
[eventpoll]
sshd      11521     root    3r  IPv4           439217     0t0       TCP
leo:ssh->114.245.94.204:61771 (ESTABLISHED)
tmux      11577     root    3u   REG              0,9       0      3786
[eventpoll]
lsof      11709     root    3r   DIR              0,3       0         1
/proc
```

打开标准错误输出的进程还真不少。看看我们是如何做到的，-d 选项加文件描述符，就是这么简单。

进程定位文件

上面的示例都是通过文件定位进程，这次我们来个逆向过程，通过进程定位文件。

还是通过示例来为大家讲解，我们尝试查看 mysqld 进程都打开了哪些文件。

第一步：查找 mysqld 的 PID。

```
[roc@roclinux ~]$ ps aux | grep mysqld
roc       5404  0.0  0.0 110372  1456 ?        S     2015   0:00 /bin/sh
/home/roc/program/mysql/bin/mysqld_safe
--defaults-file=/home/roc/program/mysql/my.cnf
--datadir=/home/roc/program/mysql/data
--pid-file=/home/roc/program/mysql/mysql.pid
roc        5619   1.0   4.2 2387556 130684 ?       Sl     2015 1555:32
/home/roc/program/mysql/bin/mysqld
--defaults-file=/home/roc/program/mysql/my.cnf
--basedir=/home/roc/program/mysql
--datadir=/home/roc/program/mysql/data
```

```
--plugin-dir=/home/roc/program/mysql/lib/plugin
--log-error=/home/roc/program/mysql/log/err.log
--pid-file=/home/roc/program/mysql/mysql.pid
--socket=/home/roc/program/mysql/tmp/mysql.sock --port=8306
1005     38892  0.0  0.0 103252   892 pts/4    S+   19:11   0:00 grep
mysqld
```

mysqld 的 PID 是 5619。

第二步：通过 PID 查找打开的文件。

```
[roc@roclinux ~]$ sudo lsof -p 5619
COMMAND  PID USER   FD   TYPE           DEVICE SIZE/OFF    NODE NAME
mysqld  5619  roc  cwd    DIR                8,17     4096  1186679
/home/roc/program/mysql-5.6.20/data
mysqld  5619  roc  rtd    DIR              8,1     4096       2 /
mysqld  5619  roc  txt    REG                8,17 87145506   927760
/home/roc/program/mysql-5.6.20/bin/mysqld
mysqld  5619  roc  mem    REG                 8,1   383504   395308
/lib64/libfreebl3.so
mysqld  5619  roc  mem    REG                 8,1  1921176   395317
/lib64/libc-2.12.so
mysqld  5619  roc  mem    REG                 8,1    90784   422586
/lib64/libgcc_s-4.4.7-20120601.so.1
mysqld  5619  roc  mem    REG                 8,1   596272   395325
/lib64/libm-2.12.so
mysqld  5619  roc  mem    REG                 8,1   987096   395723
/usr/lib64/libstdc++.so.6.0.13
mysqld  5619  roc  mem    REG                 8,1    19536   395323
/lib64/libdl-2.12.so
mysqld  5619  roc  mem    REG                 8,1    40400   395321
/lib64/libcrypt-2.12.so
mysqld  5619  roc  mem    REG                 8,1    43880   395345
/lib64/librt-2.12.so
mysqld  5619  roc  mem    REG                 8,1   142640   395341
/lib64/libpthread-2.12.so
mysqld  5619  roc  mem    REG                 8,1   154624   395305
/lib64/ld-2.12.so
mysqld  5619  roc   0r   CHR              1,3     0t0     4012 /dev/null
...
```

可以看到，我们通过-p 选项实现了这个逆向效果。我们对 lsof 的能力又有了新的认识。

查看某一用户打开的文件

如果想查找用户 getsmartoffer 在系统中都打开了哪些文件，我们可以使用-u 选项来实现：

```
# 使用选项 -u 选项指定用户
[roc@roclinux ~]# sudo lsof -u getsmartoffer
```

```
COMMAND   PID        USER   FD   TYPE            DEVICE SIZE/OFF     NODE
NAME
sshd    37126 getsmartoffer  cwd    DIR              8,1     4096        2
/
sshd    37126 getsmartoffer  rtd    DIR              8,1     4096        2
/
sshd     37126 getsmartoffer  txt    REG                8,1   525952
409162 /usr/sbin/sshd
sshd    37126 getsmartoffer  DEL    REG              0,4         31120453
/dev/zero
sshd     37126 getsmartoffer  mem    REG                8,1    18592
397322 /lib64/security/pam_limits.so
sshd     37126 getsmartoffer  mem    REG                8,1    10224
397320 /lib64/security/pam_keyinit.so
sshd     37126 getsmartoffer  mem    REG                8,1    10240
397325 /lib64/security/pam_loginuid.so
sshd     37126 getsmartoffer  mem    REG                8,1    18664
397337 /lib64/security/pam_seLinux.so
sshd     37126 getsmartoffer  mem    REG                8,1    38712
397037 /usr/lib64/libcrack.so.2.8.1
sshd     37126 getsmartoffer  mem    REG                8,1    14416
397306 /lib64/security/pam_cracklib.so
sshd     37126 getsmartoffer  mem    REG                8,1     6040
397331 /lib64/security/pam_permit.so
sshd     37126 getsmartoffer  mem    REG                8,1    10200
397324 /lib64/security/pam_localuser.so
sshd     37126 getsmartoffer  mem    REG                8,1    10208
397330 /lib64/security/pam_nologin.so
sshd     37126 getsmartoffer  mem    REG                8,1     5952
397308 /lib64/security/pam_deny.so
sshd     37126 getsmartoffer  mem    REG                8,1    14384
397342 /lib64/security/pam_succeed_if.so
sshd     37126 getsmartoffer  mem    REG                8,1    51952
397348 /lib64/security/pam_unix.so
sshd     37126 getsmartoffer  mem    REG                8,1    18592
397310 /lib64/security/pam_env.so
sshd     37126 getsmartoffer  mem    REG                8,1    14488
397339 /lib64/security/pam_sepermit.so
sshd     37126 getsmartoffer  mem    REG                8,1    65928
395333 /lib64/libnss_files-2.12.so
sshd     37126 getsmartoffer  mem    REG                8,1   240560
395808 /lib64/libnspr4.so
sshd     37126 getsmartoffer  mem    REG                8,1    14560
395810 /lib64/libplds4.so
sshd     37126 getsmartoffer  mem    REG                8,1    18720
395809 /lib64/libplc4.so
sshd     37126 getsmartoffer  mem    REG                8,1   125768
395874 /usr/lib64/libnssutil3.so
...
```

说个题外话，之所以在这里用了 getsmartoffer 这个账号，是因为我人生中的第一台外星人（Alienware）笔记本是在“美国章鱼哥”店购买的，当时正值毕业找工作，觉得他店的域名（getsmartoffer.taobao.com）很吉利，所以我的外星人笔记本也使用了这个

账号作为登录账号。哈哈。

看看哪些程序占用着端口

lsof 不仅可以根据文件找进程，还可以根据端口找进程。

对于网络端口的查找，lsof 有针对性的语法格式：

```
lsof -i[46] [protocol][@hostname|hostaddr][:service|port]
```

- 46：IPv4 or IPv6。
- protocol：TCP or UDP。
- hostname：Internet host name。
- hostaddr：IPv4 地址。
- service：服务的名字（可以不止一个）。
- port：端口号（可以不止一个）。

我们可以通过上面的这些选项来监视网络，让我们对网络状况了然于心。比如，想查看一下端口 22 上都运行着哪些程序：

```
[root@roclinux ~]# lsof -i:22
COMMAND  PID USER   FD   TYPE    DEVICE SIZE/OFF NODE NAME
sshd    1319 root    3u  IPv4     9690      0t0  TCP *:ssh (LISTEN)
sshd    1319 root    4u  IPv6     9692      0t0  TCP *:ssh (LISTEN)
sshd       5375  root       3r    IPv4  351041075          0t0    TCP
instance-l8ll9n75-2:ssh->223.72.105.30:38009 (ESTABLISHED)
sshd       5377    leo       3u    IPv4  351041075          0t0    TCP
instance-l8ll9n75-2:ssh->223.72.105.30:38009 (ESTABLISHED)
```

好了，我们和 lsof 初次见面就领略到了它的灵活和强大，而接下来的悬疑篇，我们会为大家介绍三件很玄虚又很捉摸不透的案件。

15 帮你找到幕后黑手——lsof 悬疑篇

悬案 I——消失的空间

本案讲述的是这样一个故事，在故事的开始阶段，磁盘的状态是这样的：

```
[root@roclinux ~]# df -h
Filesystem     Size  Used Avail Use% Mounted on
/dev/vda1       20G  1.3G   18G   7% /
tmpfs          1.9G     0  1.9G   0% /dev/shm
```

为了项目的需要，我们在磁盘中创建了一个很大的文件，叫作 1gb.file：

```
 [root@roclinux ~]# dd if=/dev/zero bs=1024 count=1000000 of=./1gb.file
1000000+0 records in
1000000+0 records out
1024000000 bytes (1.0 GB) copied, 3.48426 s, 294 MB/s
```

然后磁盘就变成了下面的样子，磁盘使用率从 7%一下子涨到了 12%：

```
[root@roclinux ~]# df -h
Filesystem     Size  Used Avail Use% Mounted on
/dev/vda1       20G  2.2G   17G  12% /
tmpfs          1.9G     0  1.9G   0% /dev/shm
```

为了让这个悬案看上去更真实，我们特意编写了一个小程序，来模拟业务上持续读取文件的场景：

```
 [root@roclinux ~]# cat openfilenoclose.c
#include <stdio.h>

void main() {
  FILE *file=NULL;
  char ch;
  file = fopen("1gb.file", "r");
  if (NULL==file) {
    return;
  }
  while(1) {
    ch = getc(file);
    printf("%c", ch);
  }
  fclose(file);
```

```
}
[root@roclinux ~]# ./openfilenoclose &
[1] 16184
```

几天之后，由于项目关系，我们需要更大的磁盘空间来存放其他数据，于是工程师毫不犹豫地删除了这个大文件 1gb.file，但却悲剧地发现，磁盘的可用空间并没有因此而增加：

```
#原来磁盘的使用空间为12%
[root@roclinux ~]# df -h
Filesystem     Size  Used Avail Use% Mounted on
/dev/vda1       20G  2.2G   17G  12% /
tmpfs          1.9G     0  1.9G   0% /dev/shm

#我们删除了这个大块头
[root@roclinux ~]# rm -rf 1gb.file

#文件的确没有了
[root@roclinux ~]# ls -l
total 112
-rw-r--r-- 1 root root    12 Feb 26 12:37 input.txt
-rw-r--r-- 1 root root    12 Feb 26 12:24 input.txt_bak
-rw-r--r-- 1 root root 92817 Feb 24 15:54 lsof.out
-rwxr-xr-x 1 root root  6791 Feb 26 14:49 openfilenoclose
-rw-r--r-- 1 root root   212 Feb 26 14:49 openfilenoclose.c

#可是磁盘使用空间仍然是12%，并没有减少
[root@roclinux ~]# df -h
Filesystem     Size  Used Avail Use% Mounted on
/dev/vda1       20G  2.2G   17G  12% /
tmpfs          1.9G     0  1.9G   0% /dev/shm
```

这到底是怎么回事呢？太不符合逻辑了！

为了工作顺利开展，经过仔细的调查研究，终于找到了这一悬案的真相。

原来在 Linux 系统中，rm 命令删除文件实际上只是减少了文件的 link 数，而 Linux 系统规定，只有当文件的 link 数为 0 时，文件才会真正从磁盘上被删除。另外，当进程打开了某个文件时，这个文件的 link 数就会增加 1。

知道了这个原理后，就明白为什么删除文件后磁盘的可用空间没有增加了吧。1gb.file 文件一定是被其他某个进程打开着。

我们现在的首要任务就是找到那个打开了这个文件的元凶！这个任务就交给侦探 lsof 来解决吧。

```
[root@roclinux ~]# lsof | grep 1gb.file
openfilen 16185      root    3r      REG              253,1 1024000000
```

```
266078 /root/100cmd/lsof/1gb.file (deleted)
```

真相大白于天下，原来有一个 PID 为 16185 的进程正在使用这个文件，怪不得磁盘可用空间没有得到释放呢！好吧，我们杀掉该进程之后，再来看看磁盘状态是否有变化？

```
#杀掉元凶进程
[root@roclinux ~]# kill -9 16185
[1]+  Killed                  ./openfilenoclose

#可以看到，使用空间又恢复到7%了
[root@roclinux lsof]# df
Filesystem     1K-blocks    Used Available Use% Mounted on
/dev/vda1       20641820 1285088  18308924   7% /
tmpfs            1961364       0   1961364   0% /dev/shm
```

好了，我们的第一个悬案，在 lsof 的明察秋毫下，顺利破案！

悬案 II——幕后的推手

本案的案情是这样的，Linux 系统中有一个文件，我们发现文件的大小在持续地增加。

文件大小变化趋势图如图 18 所示。

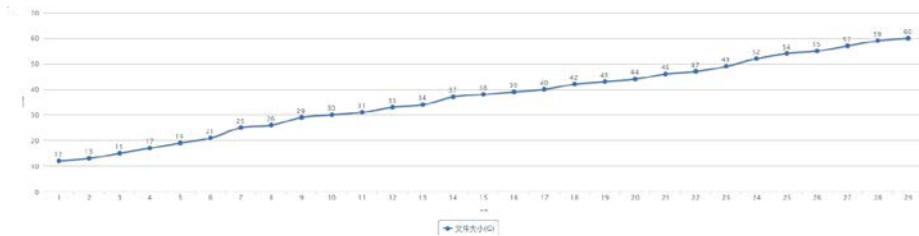


图 18　变化趋势图

大事不好，如果保持这个趋势下去的话，磁盘空间很快就会被消耗光的。这可如何是好？到底是什么程序在向文件中写数据？我们有必要立案侦查一下。

```
[root@roclinux ~]# lsof input.txt
COMMAND    PID USER   FD   TYPE DEVICE  SIZE/OFF   NODE NAME
writfile 32449 root    3w   REG  253,1 217501607 266079 input.txt
```

又是 lsof 侦探发现了线索，原来幕后的推手是进程 32449。可以先查查该程序是做什么的，如果是非法的程序，直接 kill 掉吧，消除安全隐患。

悬案 III——被占据的端口

本案是由我编写的一个程序引发的。前几天我开发了一个网络服务器程序，当我信心满满地执行程序时，却出现了下面的错误：

```
[root@roclinux ~]# ./server2
bind: Address already in use
```

追溯到我的程序代码中的错误提示：

```
if(bind(sockfd,(struct        sockaddr        *)&my_addr,sizeof(struct
sockaddr))==-1)
{
       perror("bind");
       exit(1);
}
```

这个错误的原因是 bind()在绑定服务器端口时，发现程序指定的端口已经被别的程序占用了。是被什么程序占用了呢？

首先，我们需要找出我们自己的程序中指定的端口号：

```
my_addr.sin_family = AF_INET;
my_addr.sin_port = htons(2323);
my_addr.sin_addr.s_addr = INADDR_ANY;
bzero(&(my_addr.sin_zero),8);
```

我们希望打开的端口是 2323。接下来，再次有请 lsof 帮忙查看到底是什么程序在使用该端口：

```
[root@roclinux ~]# lsof -i:2323
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
server  19994 root    3u  IPv4 581485      0t0  TCP *:3d-nfsd (LISTEN)
```

原来是 PID 为 19994 的程序呀，这是个什么程序呢？继续向下查：

```
[root@roclinux ~]# ps aux | grep 19994
root     19994  0.0  0.0   3924   420 pts/2    S    19:29   0:00 ./server
root     20016  0.0  0.0 103244   872 pts/2    S+   19:50   0:00 grep 19994
```

哦，这下清楚了，原来真正的凶手就是./server！好吧，直接关闭 ./server 程序，再来运行我们的程序，一切 OK 了。

在本文中，lsof 三立奇功，值得嘉奖。读者应该也从本文中学会了三个典型的运维问题排查过程，相信以后真的遇到这类问题，第一个想到的就是 lsof 大侦探了。

16 帮你找到幕后黑手——lsof 进阶篇

lsof 查找原理

在 lsof 悬疑篇中，我们领略了 lsof 命令的风采，高山仰止之余，有没有兴趣了解一下 lsof 命令的原理呢？

在 Linux 系统中，系统为了方便管理进程，会在/proc 下为每一个运行中的进程创建一个目录，目录名就是进程号，而在进程的目录下有一个叫作 fd 的目录，这个目录下存放的是进程打开的所有文件。而 lsof 命令搜寻的其实就是/proc/$PID/fd 下面的文件。

了解了这些知识，我们其实完全可以自己编写脚本来遍历所有进程，甚至可以实现和 lsof 一样的效果：

```
[root@roclinux ~]# for p in [0-9]*; do ls -l /proc/$p/fd; done
```

详解 lsof 输出

lsof 命令输出的信息非常丰富，下面我们就来看看这些信息的具体含义。

```
[root@roclinux ~]# lsof
COMMAND     PID      USER   FD      TYPE             DEVICE  SIZE/OFF
NODE NAME
init          1      root  cwd       DIR              253,1      4096
2 /
init          1      root  rtd       DIR              253,1      4096
2 /
init          1      root  txt       REG              253,1    150352
131833 /sbin/init
init          1      root  mem       REG              253,1     65928
12641 /lib64/libnss_files-2.12.so
init          1      root  mem       REG              253,1   1921176
2906 /lib64/libc-2.12.so
init             1        root   DEL         REG                 253,1
40 /lib64/libgcc_s-4.4.7-20120601.so.1;565c5dd4
init          1      root  mem       REG              253,1     43880
12644 /lib64/librt-2.12.so
```

```
init          1      root  mem       REG               253,1    142640
2930 /lib64/libpthread-2.12.so
init          1      root  mem       REG               253,1    265728
3897 /lib64/libdbus-1.so.3.4.0
（此处省略数百行...）
```

- COMMAND：进程的名称，如果进程的名字很长，则这里只会显示前 9 个字符。
- PID：进程标识符。如果执行命令时指定-R 参数，则父进程标识符 PPID 也会显示出来。
- USER：进程所有者。如果执行命令时指定-g 参数，则进程所属组标识符 PGID 也会显示出来。
- FD：一般是指文件描述符。
- TYPE：文件类型，如 DIR、REG 等，常见的文件类型。
  - ✧ DIR：表示目录。
  - ✧ REG：表示普通文件。
  - ✧ CHR：表示字符类型。
  - ✧ BLK：块设备类型。
  - ✧ UNIX：UNIX 域套接字。
  - ✧ FIFO：先进先出（FIFO）队列。
  - ✧ IPv4/IPv6：网际协议（IP）套接字。
- DEVICE：磁盘的名称。
- SIZE：文件的大小。
- NODE：索引节点（文件在磁盘上的标识）。
- NAME：打开文件的确切名称。

具体说说 FD

FD，在这里虽然是指文件描述符，但却可以指代两类内容：

- 第一类是文件描述符。
- 第二类是描述文件特征的标识。

先来说说第一类的文件描述符：

- 0 表示标准输入。
- 1 表示标准输出。
- 2 表示标准错误输出。

- n 表示其他文件描述符的数值。

文件描述符后面还可以跟文件状态模式和文件锁，详情如图 19 所示。

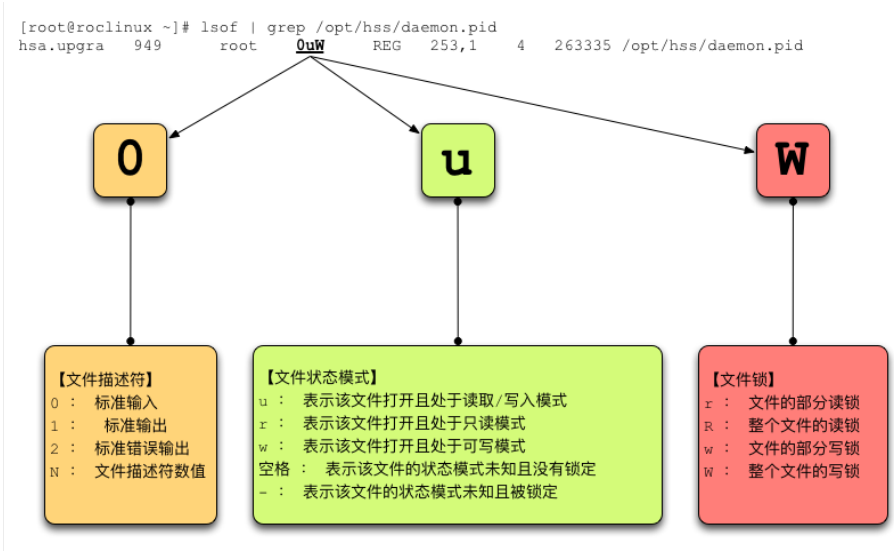


图 19　文件描述符、文件状态模式、文件锁

再说说第二类的描述文件特征的标识：

- cwd：应用程序的当前工作目录，也是该应用程序启动的目录。
- txt：该类型文件是程序代码或数据。
- mem：内存映射文件。
- pd：父目录。
- rtd：根目录。
- DEL：表示文件已经被进程删除但还在内存中存在。

好了，有关 lsof 的内容就都结束了。虽然 lsof 如此实用和高效，但还是要让大家平复一下激动的心情，因为还有一位新朋友，也非常值得大家期待，那就是下一篇的主角 fuser。

17 帮你找到幕后黑手——fuser 学习篇

符合 POSIX 标准的 fuser

前几篇文章我们见识到了 lsof 的强大功能，无论是在文件方面、网络方面，还是在侦查悬案方面，它的表现都超出了大家的预期。

但是比较遗憾的是，lsof 命令并不是 POSIX 标准中规定的命令，所以如果深究的话，lsof 命令在可移植性方面会稍微差一些。而经常拿来与 lsof 命令作比较的 fuser 命令，则属于 POSIX 标准的命令集，所以我们在本系列的最后一篇文章中，有必要和大家介绍一下 fuser 命令。

可能有人会问“POSIX”是什么，我们就在这里小科普一下。

> POSIX，是 Portable Operating System Interface 的缩写，中文称为可移植操作系统接口。它是一种有关操作系统的行业标准。POSIX 标准的职责是确保程序在源代码级别上的可移植性。
>
> 通俗地说，只要在程序开发过程中使用的是 POSIX 标准规定的 API 和命令，那么这个程序就应该可以在任何其他符合 POSIX 标准的操作系统上编译和执行。
>
> 而 Linux 操作系统做到了与 POSIX 标准的兼容，可以这样说，Linux 是完全遵循 POSIX 标准的。

科普之后，大家应该能够理解 lsof 和 fuser 在可移植性方面的区别了吧！

fuser 简单用法

fuser 命令在功能上和 lsof 命令有很多相似之处，它可以显示出磁盘上的文件、目录，甚至网络端口正在被什么程序使用，并可以展示出这些程序的详细信息。

我们先来看一个示例。

```
[root@roclinux ~]# fuser -v /home/roc
```

```
                     USER        PID ACCESS COMMAND
/home/roc:             roc      12384 ..c.. tmux
```

上述命令列出了所有正在打开/home/roc/这个目录（不含目录下的文件）的进程。其中添加-v 选项后，会输出下列几类丰富的信息内容：

- USER：进程所属的用户。
- PID：进程的 ID。
- COMMAND：进程程序名。
- ACCESS：表示访问关系。

其中，ACCESS 表示 fuser 找出的进程和指定文件的访问关系，具体这些关系有哪些，我们会单独抽出一个段落来介绍。

fuser 输出中的 ACCESS

ACCESS 表示的访问关系有哪些呢？在介绍这个问题之前，我们先来看看下面的示例。

```
[root@roclinux ~]# fuser -v /home/roc
                     USER        PID ACCESS COMMAND
/home/roc:             roc      12384 ..c.. tmux
```

上面示例显示出了进程 tmux 和/home/roc 目录的访问关系是 c。c 表示“将此文件作为当前目录使用”，也就是说，tmux 进程把/home/roc 作为当前目录使用。除了 c 外，还有其他几种访问关系：

- `c`：将此文件作为当前目录使用。
- `e`：将此文件作为可执行对象使用。
- `r`：将此文件作为根目录使用。
- `s`：将此文件作为共享库（或其他可装载对象）使用。
- `m`：将此文件作为映射文件或者共享库。
- `f`：打开此文件。默认不显示。
- `F`：打开此文件，用于写操作。默认不显示。

注意：上面的访问关系中，f 和 F 比较特殊，它们只有在使用选项-v 的情况下才会显示，其他情况不会显示。

```
# 使用fuser显示打开当前目录的进程
# 输出12384c， 12384表示进程PID，而c就是ACCESS中的c
[roc@roclinux ~]$ fuser .
.:                   12384c

# 使用选项-v，我们也可以清楚地看到ACCESS中的c
[roc@roclinux ~]$ fuser -v .
                     USER       PID ACCESS COMMAND
.:                   roc       12384 ..c.. tmux

# 下面来看看ACCESS中的F的显示

# 终端中打开文件11.txt
[roc@roclinux ~]$ vim 11.txt

# 新打开一个终端，执行fuser -v
# 指定文件为vim临时文件.11.txt.swp（注意这是使用vim的一个坑，我们在lsof中介绍过）
# 这里可以看到ACCESS的是F
[roc@roclinux ~]$ fuser -v .11.txt.swp
                     USER       PID ACCESS COMMAND
.11.txt.swp:           root     21575 F.... vim

# 如果没有选项v
# 则输出 21575
# 我们看到21575后没有任何类型。F未显示
[roc@roclinux ~]$ fuser .11.txt.swp
.11.txt.swp:          21575
```

请正确理解-m 选项

除了像上例那样使用 fuser 之外，我们还经常使用 fuser 的-m 选项。在这里，我们就为大家介绍下-m 选项的作用和用法。

```
#和上例的命令是一样的
[root@roclinux ~]# fuser -v /home/roc
                     USER       PID ACCESS COMMAND
/home/roc:             roc      12384 ..c.. tmux


#注意，这里使用了-m选项，明显输出了更多的内容
[root@roclinux ~]# fuser -v -m /home/roc
                     USER       PID ACCESS COMMAND
/home/roc:             root         1 .rce. init
                     root         2 .rc.. kthreadd
                     root         3 .rc.. migration/0
                     root         4 .rc.. ksoftirqd/0
（此处省略数十行）
                     roc      12384 .rce. tmux
                     root     29062 .rce. su
```

```
                    root     29063 .rce. bash
```

不卖关子，-m 选项表示查阅有多少程序正在使用某个目录底下的文件系统（目录下的所有子目录和文件）。因此，在上面的示例中，我们看到多出来的大量进程，都是正在使用/home/roc 目录下的文件的进程。

通过端口定位进程

其实 fuser 比你想象的要强大得多，fuser 不仅能找到正在使用文件、目录的是哪些进程，还可以定位到占用了某个网络端口的是哪个进程。

比如，你想看看都有哪些程序占用了本机的 22 端口，则可以通过下面的命令来实现：

```
[root@roclinux ~]# fuser -v -n tcp 22
                    USER       PID ACCESS COMMAND
22/tcp:              root      1319 F.... sshd
                    root      6358 f.... sshd
                    roc       6360 F.... sshd
```

其实还有另一种写法，也是可以的：

```
[root@roclinux ~]# fuser -v 22/tcp
                    USER       PID ACCESS COMMAND
22/tcp:              root      1319 F.... sshd
                    root      6358 f.... sshd
                    roc       6360 F.... sshd
```

也就是说，上面的-n tcp22 可以简写为 22/tcp 的形式。这里的 tcp 是协议类型。fuser 中的协议类型有 file（默认）、tcp 和 udp 三种。你可以自由组合它们来查找你想要的信息。

杀死进程一步走

当我们希望杀掉和一个文件关联的进程时，我们通常会采取这样的操作步骤：

1. 通过 lsof 或 fuser 找到都有哪些进程正在打开此文件。
2. 记录好这些进程的 PID。
3. 调用 kill 命令向这些 PID 发送 SIGKILL 信号。

这三步都需要大家手动来完成，是不是觉得有些烦琐呢？

一款好的产品或服务，是会为用户提供超出预期的体验的，fuser 命令就是如此，

它为我们提供了-k 选项，可以一步完成上面的三步：

```
[root@roclinux ~]# fuser -v -k /home/roc
```

方法虽好，但可不要滥用。同学们一定要谨慎使用-k 选项。毕竟它的杀伤力还是很大的，万一杀掉了像 sshd 这样的进程，你可就无法登录到你的服务器了！

fuser 和 lsof 大比对

通过这几篇文章的介绍，可以看出 fuser 和 lsof 在实现功能上还是比较类似的，只是一些细节上有点差别。下面就来比较一下，如表 18 所示。

表 18　lsof 和 fuser 功能对比

对比项	lsof	fuser
思路	通过进程找文件	通过文件找进程
所属标准	-	POSIX
接收参数类型	文件、PID、网络端口	文件、网络端口
进程的输出	进程的详细信息	进程的 ID
是否可发送信号	不可以	可以（-k 或-signal 发送信号）

本文我们介绍了 lsof 和 fuser 的知识，还带着大家侦破了三桩悬案，都掌握了么？相信在未来的日子里，大家也都可以独立办案了！

18 ps 命令看着简单，其实很难

最简单的 ps 用法

本文的主角，不是那款大名鼎鼎的图像设计软件 Photoshop，而是 Linux 系统中的进程状态查看命令 ps，即 Process Status。

如果你想查看你的服务器上有哪些进程，这些进程属于哪些用户，这些进程消耗了多少 CPU 资源，这些进程占据了多少内存资源，那么就快来一起学习 ps 命令吧。

想必大家已经猜到了，最简单的 ps 用法，就是直接输入 ps，按回车。

不带任何参数的 ps 命令，会显示当前用户在当前终端下的进程信息，而不会显示其他终端的进程信息，例如：

```
[root@roclinux ~]# ps
  PID TTY          TIME CMD
18006 pts/2    00:00:00 bash
18041 pts/2    00:00:00 ps
```

可以看到，ps 显示出了 4 列内容，它们的含义如表 19 所示。

表 19　ps 显示的 4 列内容的含义

列　名	内　容
PID	进程 ID
TTY	启动此进程的终端名称
TIME	进程所占用的 CPU 时间总和，单位为秒
CMD	进程命令

好了，恭喜你，你已经掌握了 ps 的最简单用法啦。

不过，还请不要骄傲，ps 命令绝对是一个看着简单但其实很难的命令，后面还有很多内容等着大家去学习呢。

最常用的 ps 用法

刚才已经掌握了 ps 最简单的用法，下面我们就从实战出发，来学习 ps 的最常用用法。

相信每个人的使用习惯都存在差异，所以我们很难精准统计大家都是怎么使用 ps 命令的，但是从我的观察和经验来看，查看系统中全部进程信息，应该是最为常见的场景了。

```
#我们使用了aux组合选项
[root@roclinux ~]# ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  19224   608 ?        Ss    2015   0:13 /sbin/init
root         2  0.0  0.0      0     0 ?        S     2015   0:00 [kthreadd]
root           3   0.0   0.0        0       0  ?          S       2015    0:01
[migration/0]
root           4   0.0   0.0        0       0  ?          S       2015    0:06
[ksoftirqd/0]
root           5   0.0   0.0        0       0  ?          S       2015    0:00
[migration/0]
root         6  0.0  0.0      0     0 ?        S     2015   0:00 [watchdog/0]
（此处省略数十行...）
roc      35631  0.0  0.1  28160  3400 ?        Rs   Jan27  10:53 tmux new -s
roc
roc      35722  0.0  0.1 116544  4972 pts/3    Ss+  Jan27   0:02 -bash
wdcpu     38661   0.0   0.1  112364   6152  ?          S      Feb16    0:05
/www/wdLinux/wdapache/bin/httpd
roc        41039   0.0  19.7  1862340  613460  ?        Sl    Jan18   47:58
/usr/local/rvm/rubies/ruby-2.2.3/bin/ruby /usr/local/rvm/gems/rub
roc      43585  0.0  0.1 116648  6076 pts/1    Ss+  Mar09   0:01 -bash
（此处省略数十行...）
```

是的，ps aux 便是我们最常用的 ps 用法了。它可以展示出系统中所有进程的状态信息，非常便于管理员进行全局查看。

那么，aux 到底代表什么含义呢？

- a：显示各终端上的所有进程。
- u：会展示进程所属用户名。
- x：对于没有关联到终端上的进程，也展示出来。

你知道 aux 和-aux 的区别么

遥想毕业当年，参加面试时，面试官提出了“aux 和-aux 的区别”这个问题，而涉世未深、年幼稚嫩的我，并没有回答得很好，于是我下定决心要把这个知识点研

究透彻。

相信大家在未来的笔试面试中，也很有可能会遇到这个问题，所以，我们就在本书，为大家揭晓谜题！

为了有一个直观的感受，我们先来看看 aux 和-aux 输出的差异，先玩一次“大家来找茬”：

```
[root@roclinux ~]# ps aux|head -n 5
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  19224   608 ?        Ss    2015   0:13 /sbin/init
root         2  0.0  0.0      0     0 ?        S     2015   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S     2015   0:01
[migration/0]
root         4  0.0  0.0      0     0 ?        S     2015   0:06
[ksoftirqd/0]
（此处省略数十行）

[root@roclinux ~]# ps -aux|head -n 5
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  19224   608 ?        Ss    2015   0:13 /sbin/init
root         2  0.0  0.0      0     0 ?        S     2015   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S     2015   0:01
[migration/0]
root         4  0.0  0.0      0     0 ?        S     2015   0:06
[ksoftirqd/0]
（此处省略数十行）
```

发现区别了么？即便你眼尖，也不会发现区别的，因为真的是一模一样，根本没有区别。

那么，aux 和-aux 区别到底是什么呢？

ps -aux，其实应该理解为：

```
#-aux的拆解形式
[root@roclinux ~]# ps -a -u x
```

它的准确解释是：

- -a 部分：显示所有当前终端的所有进程。
- -u x 部分：显示用户名为“x”的用户的所有进程。

假如系统中不存在用户名为“x”的用户的话，ps 就会把“ps -aux”解释为“ps aux”，这也就能解释为什么上例中 ps aux 和 ps -aux 的输出一模一样的原因了。

有同学会问，为什么会有这样的处理逻辑呢？为什么要有 aux 和-aux 两种不同的表达格式呢？其实这是有历史原因的，下面，我们就带大家进入这段“鲜为人知”

的历史故事。

ps 的三格式之殇

ps 命令，是一个有故事的命令，由于历史原因，它其实拥有三种不同的书写格式：

- BSD 格式：选项前不需要加短横线-，几个选项可以组合在一起使用，如 ps aux。
- UNIX 格式：选项前加短横线-，几个选项可以组合在一起使用，如 ps -aux。
- GNU 长格式：选项前加双短横线--，如 ps --format。

在 ps 命令中，我们常用的格式是 BSD 格式和 UNIX 格式，两种格式没有好坏之分，只有帮派之别。

而我们想查看的信息，无论是通过 BSD 格式，还是 UNIX 格式，基本都可以实现，只是在信息丰富度上略有差异，如下面的例子。

```
#BSD格式
[root@roclinux ~]# ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  19224   608 ?        Ss   2015   0:13 /sbin/init
root         2  0.0  0.0      0     0 ?        S    2015   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    2015   0:01
[migration/0]
root         4  0.0  0.0      0     0 ?        S    2015   0:06
[ksoftirqd/0]
（此处省略数十行）

#同样效果，可以通过UNIX格式实现
[root@roclinux ~]# ps -eF
UID        PID  PPID  C    SZ   RSS PSR STIME TTY          TIME CMD
root         1     0  0  4806   608   3  2015 ?       00:00:13 /sbin/init
root         2     0  0     0     0   2  2015 ?       00:00:00 [kthreadd]
root         3     2  0     0     0   0  2015 ?       00:00:01 [migration/0]
root         4     2  0     0     0   0  2015 ?       00:00:06 [ksoftirqd/0]
（此处省略数十行）
```

可以看到，ps aux 和 ps -eF 展示的进程信息基本类似，那么-e 和-F 的作用是什么呢？

- -e 选项：显示全部的进程，而不仅仅是当前终端下的进程，还包括没有在终端运行的进程。
- -F 选项：显示详尽的进程信息。

通过上面的例子我们看到，大家只需选用自己熟悉的命令格式即可，哪种格式用

得顺手就用哪种，不用苛求一定要采用哪一种而去否定另一种。

我来控制展示哪些列

通过前面的学习我们知道，不带任何参数的 ps 命令会打印 4 列进程信息，而使用-F 会显示更多更丰富的列信息。如果觉得 4 列太少而-F 又展示得太多，那么就需要我们自己来自定义列信息了，这时我们要用到-o 选项：

```
[root@roclinux ~]# ps -eo pid,user,cmd,start
  PID USER     CMD                         STARTED
    1 root     /sbin/init                   Nov 13
    2 root     [kthreadd]                   Nov 13
    3 root     [ksoftirqd/0]                 Nov 13
    5 root     [kworker/0:0H]                Nov 13
    7 root     [rcu_sched]                  Nov 13
```

如果你想修改默认列名的名称，也是可以的：

```
[root@roclinux ~]# ps -eo pid,user,cmd=COMMAND,start
  PID USER     COMMAND                     STARTED
    1 root     /sbin/init                   Nov 13
    2 root     [kthreadd]                   Nov 13
    3 root     [ksoftirqd/0]                 Nov 13
    5 root     [kworker/0:0H]                Nov 13
    7 root     [rcu_sched]                  Nov 13
```

ps 给予了我们充分的信任，可选的列名非常多，我们在这里直接列出来，大家也可以通过 man 来查看到：

```
%cpu、%mem、args、blocked、bsdstart、bsdtime、c、caught、cgroup、class、
cls、cmd、comm、command、cp、cputime、egid、egroup、eip、esp、etime、euid、
euser、f、fgid、fgroup、flag、flags、fname、fuid、fuser、gid、group、ignored、
label、lstart、lwp、ni、nice、nlwp、nwchan、pcpu、pending、pgid、pgrp、pid、
pmem、policy、ppid、psr、rgid、rgroup、rip、rsp、rss、rssize、rsz、rtprio、
ruid、ruser、s、sched、sess、sgi_p、sgid、sgroup、sid、sig、sigcatch、sigignore、
sigmask、size、spid、stackp、start、start_time、stat、state、suid、suser、
svgid、svuid、sz、thcount、tid、time、tname、tpgid、tt、tty、ucmd、ucomm、
uid、uname、user、vsize、vsz、wchan
```

足有 98 个之多，想要完全了解和掌握这些列名的含义，并非易事，是不是再一次感受到 ps 命令并不是一个容易深入掌握的命令了呢？

ps 输出列信息汇总

ps 能够输出的列信息非常多，有时候会看得眼花。不同的命令选项输出的列信息

不太一样，我们在这里整理了一些常用列信息，以供查阅，如表 20 所示。

表 20　ps 命令的常用列信息

列　　名	内　　容
%CPU	进程所占用的 CPU 时间占比
%MEM	进程所使用的物理内存占比
ADDR	进程的内存地址
C 或 CP	CPU 利用率
CMD 或 COMMAND	进程名及其参数
NI	进程的 NICE 值，用于调节优先级
F	进程的旗标，又称信号量，用于进程互斥加锁等
PID	进程编号
PPID	进程的父进程编号
PRI	进程优先级
RSS	实际内存使用量，单位是 KB
S 或 STAT	进程的状态（S 代表休眠，R 代表可运行，Z 代表僵尸进程，T 代表停止，0 代表正在运行）
START 或 STIME	进程的开始时间
SZ	虚拟内存使用量
TIME	进程所占用的 CPU 时间总和，单位为秒
TT 或 TTY	启动此进程的终端名称
UID 或 USER	进程所有者的用户 ID，以及 USER 所对应的用户 ID
WCHAN	如果此进程正在睡眠，则显示睡眠中的系统调用名

筛选并聚焦

ps 命令提供了不少用来筛选冗余信息的选项，以便用户可以聚焦到所需信息。下面我们就来一起看看这些实用的选项。

当我们想查看指定用户进程的状态的时候，比如我们要查看用户 root 的进程，则可以通过-u 选项来过滤 ps 的返回结果。

```
[root@roclinux ~]# ps -u root
  PID TTY          TIME CMD
    1 ?        00:00:01 init
    2 ?        00:00:00 kthreadd
    3 ?        00:00:07 ksoftirqd/0
    5 ?        00:00:00 kworker/0:0H
    7 ?        00:04:46 rcu_sched
    （后面省略数十行）
```

再看另一个选项，当系统管理员想要查看由 root 用户运行的进程和这个进程的其他相关信息时，可以通过下面的命令实现。

```
#这个命令是不是很怪，下面会解释的
[root@roclinux ~]# ps -U root -u root u
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.3  33748  1964 ?        Ss   2015   0:01 /sbin/init
root         2  0.0  0.0      0     0 ?        S    2015   0:00 [kthreadd]
root         3  0.0  0.0       0      0 ?          S      2015    0:07
[ksoftirqd/0]
root         5  0.0  0.0       0      0 ?          S<     2015    0:00
[kworker/0:0H]
root         7  0.0  0.0      0     0 ?        S    2015   4:46 [rcu_sched]
```

你知道上面命令出现的几个 u/U 是什么含义么：

- -U 选项：按实际用户 ID（RUID）筛选进程。
- -u 选项：按有效用户 ID（EUID）筛选进程。
- u：按用户名和进程号的顺序来显示进程，其输出将由 User、PID、%CPU、%MEM、VSZ、RSS、TTY、STAT、START、TIME 和 COMMAND 这几列组成。

除了按用户聚焦，我们还可以按命令名聚焦。使用-C 选项，后面加上你要找的命令名称，就可以只显示该命令的相关进程。比如想显示名为 getty 的进程的信息，就可以使用下面的命令。

```
[root@roclinux ~]# ps -C getty
  PID TTY          TIME CMD
24519 tty4     00:00:00 getty
24521 tty5     00:00:00 getty
24522 tty2     00:00:00 getty
24523 tty3     00:00:00 getty
24524 tty6     00:00:00 getty
24525 tty1     00:00:00 getty
```

如果你的程序跑得特别慢，则可能是因为 CPU 或者内存被耗尽的缘故。aux 选项可以让你根据 CPU 或者内存的使用量来查找异常占用资源的进程。不过默认情况下，ps 是按照 PID 排序的，我们可以通过--sort 选项来设置对结果的排序方法，比如：

- -pcpu：表示按 CPU 使用率进行降序排序。
- +pcpu：表示按 CPU 使用率进行升序排序。
- -pmem：表示按内存进行降序排序。

下面来一起看看示例，猜猜它们都是按照什么顺序展示的：

```
[root@roclinux ~]# ps aux --sort -pcpu
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.3  33748  1964 ?        Ss   2015   0:01 /sbin/init
root         2  0.0  0.0      0     0 ?        S    2015   0:00 [kthreadd]
root            3   0.0   0.0        0      0 ?          S      2015    0:07
[ksoftirqd/0]
root            5   0.0   0.0        0      0 ?          S<     2015    0:00
[kworker/0:0H]
root         7  0.0  0.0      0     0 ?        S    2015   4:46 [rcu_sched]
（此处省略数十行...）

[root@roclinux ~]# ps aux --sort=+pcpu,-pmem
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
syslog     397  0.0  1.1 260068  5616 ?        Ssl  Nov13   0:00 rsyslogd
root     11599  0.0  0.8 103768  4252 ?           Ss   11:08   0:00 sshd:
root@pts/1
root     11638  0.0  0.8 103768  4252 ?           Ss   11:23   0:00 sshd:
root@pts/3
root     11523  0.0  0.8 103768  4248 ?           Ss   09:58   0:00 sshd:
root@pts/0
（此处省略数十行...）
```

排序条件还可以进行组合，上面第二条命令是按照 CPU 升序、内存降序进行排列的。

ps 还能查线程呢

如果你想查看特定进程的线程信息，则可以使用-L 选项，后面加上特定的 PID 就可以啦。如果使用-T 选项，还会显示其 SPID。

```
[root@roclinux ~]# ps -ef | grep mysql
leo             10673  10523    0  Mar15  ?                 00:01:35
/home/leo/program/mysql-5.7.11/bin/mysqld
root     28951 12442  0 15:08 pts/2    00:00:00 grep mysql

[root@roclinux ~]# ps -L 10673
  PID   LWP TTY      STAT   TIME COMMAND
10673    10673    ?                              Sl                  0:00
/home/leo/program/mysql-5.7.11/bin/mysqld
10673    10674    ?                              Sl                  0:00
/home/leo/program/mysql-5.7.11/bin/mysqld
10673    10675    ?                              Sl                  0:00
/home/leo/program/mysql-5.7.11/bin/mysqld
10673    10676    ?                              Sl                  0:00
/home/leo/program/mysql-5.7.11/bin/mysqld
10673    10677    ?                              Sl                  0:00
/home/leo/program/mysql-5.7.11/bin/mysqld
10673    10678    ?                              Sl                  0:00
/home/leo/program/mysql-5.7.11/bin/mysqld

[root@roclinux ~]# ps -T 10673
```

```
  PID  SPID TTY      STAT   TIME COMMAND
10673    10673    ?                          Sl                 0:00
/home/leo/program/mysql-5.7.11/bin/mysqld
10673    10674    ?                          Sl                 0:00
/home/leo/program/mysql-5.7.11/bin/mysqld
10673    10675    ?                          Sl                 0:00
/home/leo/program/mysql-5.7.11/bin/mysqld
10673    10676    ?                          Sl                 0:00
/home/leo/program/mysql-5.7.11/bin/mysqld
10673    10677    ?                          Sl                 0:00
/home/leo/program/mysql-5.7.11/bin/mysqld
10673    10678    ?                          Sl                 0:00
/home/leo/program/mysql-5.7.11/bin/mysqld
```

这里面有两个名词，要为入门的同学科普一下，否则在这个段落会留下疑惑的：

- LWP，是 Light Weight Process 的缩写，中文叫作轻量级进程，我们就认为它是用户线程就好了。
- SPID，表示的是系统中的线程 ID。

神奇的进程树

有时候我们希望以树形结构显示进程，即展示子进程和父进程的关系，可以使用 BSD 格式的 axjf 选项来实现。

```
[root@roclinux ~]# ps axjf
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
    0     2     0     0 ?           -1 S       0   0:00 [kthreadd]
    2     3     0     0 ?           -1 S       0   0:07  \_ [ksoftirqd/0]
    2     5     0     0 ?           -1 S<      0   0:00  \_ [kworker/0:0H]
    2     7     0     0 ?           -1 S       0   4:46  \_ [rcu_sched]
    （此处省略数十行...）
```

axjf 四个选项的含义，解释如下：

- a：显示终端上的所有进程，包括其他用户的进程。
- x：显示没有控制终端的进程。
- j：用任务格式来显示进程。
- f：用 ASCII 字符显示树状结构，表达程序间的相互关系。

其实，我们也可以使用 UNIX 格式来实现，此时就应该使用-ejH 选项来实现。

```
[root@roclinux ~]# ps -ejH
  PID  PGID   SID TTY          TIME CMD
    2     0     0 ?        00:00:00 kthreadd
    3     0     0 ?        00:00:00   migration/0
    4     0     0 ?        00:00:03   ksoftirqd/0
    （此处省略数十行...）
```

-ejH 选项的含义如下：

- -e：显示终端上的所有进程，包括其他用户的进程。
- -j：用任务格式来显示进程。
- -H：显示树状结构，表示程序间的相互关系。

可以看出，-ejH 方法是通过 CMD 列的缩进来表达父子进程关系的，视觉上不如 axjf 线段表示法直观。

好了，ps 命令的知识我们暂且介绍到这里。ps 是一个“只有两个字母、但它的知识可以写一本书”的命令，看着简单其实很难，希望大家多用 ps 命令，用到老学到老！

19 kill，这个杀手不太冷

Kill 的弹夹

在 Linux 的世界里，每一个后台进程都好像是一个特工，默默地执行着派发给它们的秘密任务。一旦总部发现某位特工遇到了状况，就会派出我们的 007，也就是本文的主角 kill，来协助其解决问题。和 007 不同的是，kill 这位杀手不是亲自上阵杀敌，而是向进程发送特定的信号，让进程根据信号来自行解决。

首先，我们使用-l 选项，来看看 kill 这位杀手都有哪些类型的子弹。

```
#使用-l选项，可以查看kill可以发出哪些信号
[root@roclinux ~]# kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      34) SIGRTMIN
35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4
39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

放眼望去，足有 64 种信号，想当好一名杀手，也并非易事啊！

如果在-l 选项后加上参数，还可以查看对应的信号编号或名称：

```
[root@roclinux ~]# kill -l SIGKILL
9
[root@roclinux ~]# kill -l KILL
9
[root@roclinux ~]# kill -l 9
KILL
```

信号种类虽然多，但常用的信号并不多，如表 21 所示。

表 21　常用的信号及说明

信号名称	信号编号	说　　明
HUP	1	终端断线
INT	2	中断（同 Ctrl+C），相当于按下 Ctrl+C 组合键来结束前台进程
QUIT	3	退出（同组合键 Ctrl+\）
TERM	15	终止
KILL	9	强制终止
CONT	18	继续（与 STOP 相反， fg/bg 命令）
STOP	19	暂停（同组合键 Ctrl+Z）

学习正确的射击方法

本书是偏向于用示例来讲解命令格式，因为直接讲解命令格式太枯燥乏味了。不过对于 kill，这个担心则不存在，因为 kill 的命令格式相当简单，学习起来并不会感到枯燥。

kill 的命令格式为：

```
kill [选项] [进程号]
```

kill 的常用选项并不多：

- -l 选项：列出全部信号类型。如果在-l 后加上信号名称，则可以查看对应的编号，反之亦然。
- -s 选项：可以指定要发出的信号，等同于“-信号”，向目标进程发送指定的信号类型。
- 无选项：则会向目标进程发送默认的终止信号，即 SIGTERM 信号，编号为 15。

kill 的进程号也是很有讲究的：

- 当进程号大于 0 时：则是向目标进程发送信号，此处的进程号可以是几个进程号的集合，之间用空格分开。
- 当进程号等于 0 时：则是向当前进程组的所有进程发送信号。
- 当进程号等于-1 时：则是向除当前 kill 进程和 init 进程之外的所有进程发送信号。

- 当进程号小于-1 时：比如是-23456，则是向进程组 PGID 为 23456 的所有进程发送信号。

看了这些，相信大家已经有了直观的感受，接下来我们马上进入紧张刺激的实战演习吧！

扣下扳机前的瞄准

在磨枪霍霍的时候，你需要定位目标进程的进程号，做到有的放矢。在叩响扳机之前，杀手需要使用 ps、jobs、pidof、pstree、top 等命令来确定进程号，才能准确地瞄准和射击。

如果我想把系统中运行 python 脚本的进程都杀掉，应该怎么做呢？

```
#通过ps命令找到PID
[root@roclinux ~]# ps -ef |grep python
root     23487 23471  5 10:30 pts/3    00:00:00 python main.py
root     23494 23408  0 10:30 pts/1    00:00:00 grep --color=auto python

#瞄准并射击
[root@roclinux ~]# kill 23487

#再次瞄准并射击，发现目标猎物已经不存在了
[root@roclinux ~]# kill 23487
-bash: kill: (23487) - No such process

#再次通过ps命令查看的确不存在了
[root@roclinux ~]# ps -ef |grep python
root     23496 23408  0 10:31 pts/1    00:00:00 grep --color=auto python
```

在本例中，我们使用了不带信号类型的 kill，于是 SIGTERM 信号会被发送给 23487 号进程，潜台词是让目标进程释放资源并退出。当然，如果进程 23487 事先设置了忽略 SIGTERM 信号，那么进程的行为可能就不会如我们所愿了。

传说中的 9 号子弹

江湖上所说的“9 号子弹”，就是 SIGKILL 信号，它是一个几乎无法被忽略的特殊信号，几乎任何进程接收到 9 号子弹之时，都是自杀之日，9 号子弹从未失手。

发射 9 号子弹的方法也有很多，下面四种都是正确的方法：

```
kill -9 12345
kill -KILL 12345
kill -SIGKILL 12345
```

```
kill -s 9 12345
```

如果你以为 9 号子弹可以“人挡杀人、佛挡杀佛”，那你就错了。世上还是有能降服 9 号子弹的神仙的，那就是进程号为 1 的 init 进程，它可谓是 Linux 世界的造物主，它是 Linux 内核启动的第一个进程，9 号子弹在 init 面前也只能低下高昂的头颅。

我们来看看当 9 号子弹遇到 1 号 init 进程时的情景：

```
#init进程的PID为1
[root@roclinux ~]# ps -ef|grep init
root        1     0  0  2015 ?       00:00:01 /sbin/init
root    23548 23471  0 12:04 pts/3   00:00:00 grep --color=auto init

#向1号进程发射9号子弹
[root@roclinux ~]# kill -9 1

#1号进程岿然不动
[root@roclinux ~]# ps -ef|grep init
root        1     0  0  2015 ?       00:00:01 /sbin/init
root    23550 23471  0 12:04 pts/3   00:00:00 grep --color=auto init
```

果然，init 进程是 Linux 世界里唯一对 9 号子弹免疫的进程。这或许是 9 号子弹一生中的唯一遗憾吧。

9 号子弹的副作用

9 号子弹能强行终止进程，但也会带来一些副作用，这是大家要提前了解的。

与其他信号不同的是，SIGKILL 信号（也就是 9 号子弹的官方称谓）并没有通知目标进程让其进行自我清理，而是在进程运行中冷不防地让它突然中止，这可能会造成系统资源无法正常释放、数据无法同步到磁盘等情况。所以，在使用 SIGKILL 信号前一定要三思。

0 号子弹

通过 kill -l 可以查看 kill 可以使用的所有信号，但是并没有编号为 0 的信号，标题里所说的 0 号子弹，又是何物呢？

我们通过-l 选项看看对 0 号子弹的解释：

```
[roc@roclinux ~]$ kill -l 0
T
```

kill -l 返回值是 T，也就是 Test 的意思，所以信号 0 其实是一个测试信号。

那-0 能做哪些测试呢？它可以测试“目标进程是否存在”。我们可以用下面的方法来达到测试的效果：

```
#测试PID为12345的进程是否存在
[roc@roclinux ~]$ kill -0 12345
-bash: kill: (12345) - No such process
```

除了测试进程是否存在之外，0 号子弹还可以测试当前用户是否拥有向目标进程发信号的权力。

来看下面的例子。

```
#锁定目标
[roc@roclinux ~]$ ps -ef |grep python
root     23701 23685  0 14:13 pts/6    00:00:00 python main.py
idcuser  23722 23716  0 14:15 pts/4    00:00:00 grep python

#发射0号子弹
[roc@roclinux ~]$  kill -0 23701
-bash: kill: (23701) - Operation not permitted
```

用户 roc 想测试是否能够杀掉拥有者为 root 的进程 23701，结果却提示没有权限（Operation not permitted）。

终结后台作业

kill 命令还支持杀掉后台作业，命令格式如下：

```
kill -信号类型 %后台工作编号
```

下面我们将工作编号为 1 的进程杀掉：

```
[root@roclinux ~]# ping www.baidu.com > output.txt  &
[1] 24749

[root@roclinux ~]# jobs
[1]+  Running                 ping www.baidu.com > output.txt &

#杀掉后台作业
[root@roclinux ~]# kill -9 %1

[root@roclinux ~]# jobs
[1]+  Killed                  ping www.baidu.com > output.txt
```

好了，有关 kill 的知识就全部介绍完了，大家也了解到了江湖上流传的 kill 杀手和 9 号子弹的故事。和这样一位杀手为伴，是不是既兴奋又紧张呢？

20 作业控制命令一览

到底什么是作业控制呢

在 Linux 的世界里，作业是由一个或多个相关进程所组成的，用户可以运行多个作业，并且在作业之间进行切换。

而作业控制，指的是对作业的行为进行控制，允许用户对作业进行前后台的切换和终止操作。

麻省理工学院的课程中，也有对作业控制的描述，原文是这样表述的：“Job control refers to the ability to selectively stop (suspend) the execution of processes and continue (resume) their execution at a later point. ”

要想学好作业控制，则需要掌握的命令如下：

- 后台（&）命令：让作业在后台运行。
- Ctrl+Z 快捷键：让作业转到后台并停止。
- jobs 命令：列出当前作业列表。
- fg 命令：将一个作业切换到前台并运行。
- bg 命令：将一个作业切换到后台并运行。
- kill 命令：终止一个作业。

说说前台后台

当你在终端中正常启动一个作业时，它会默认在前台运行。前台运行的作业的共同特点是它会从标准输入来接收用户的指令，同时通过标准输出将返回信息展示给用户。

而在后台运行的作业呢，则脱离了标准输入和标准输出，并不会直接和用户打交道，而是自己在用户看不到的地方默默地运行。

其实 Windows 的图形化窗口界面，就是非常典型的前后台切换效果。你点击哪个

窗口，哪个窗口就切换到前台，而其他窗口则自觉地切换到后台。

后台符让你的程序隐居山林

后台符，即&符号，只要你把它放在要执行的命令的后面，按下回车键后，你就会发现你要执行的命令乖乖地隐居后台，无影无踪了。

假如一个命令会运行很长一段时间。我们建议你把它放在后台运行，因为我们都不想耗费时间观看一个命令的运行过程，我们还有很多其他事情要做呢。

```
$ sleep 2000 &

[1] 23336
```

注释：在所要执行的命令后面加上空格，再加上&符号即可实现后台执行。命令所返回的“[1]”表示的是你放到后台的这个任务的任务编号（Job ID）。而 23336 则是这个任务对应的进程的进程 ID。

就这么简单，sleep 已经默默地到后台去工作了。你可以继续运行其他命令啦，完全不用照顾 sleep 的情绪。

此处，需要特别说明一下，如果我们的作业是由一组命令组成的，那么放到后台时，所返回的进程 ID 是哪一个呢，你可以先猜猜，我们通过一个例子来揭晓答案：

```
#我们将一个由多个进程组成的作业，放置到后台运行
[roc@roclinux ~]$ ./deng | ./denga | ./dengb &
[1] 33364

#通过jobs查看任务列表
[roc@roclinux ~]$ jobs
[1]+  Running                 ./deng | ./denga | ./dengb &

#通过查看进程ID，可以发现，命令所返回的进程ID是命令中最后一个程序的进程ID
[roc@roclinux ~]$ ps auxww|grep -i deng
roc      33362  0.0  0.0   3920   368 pts/0    S    00:14   0:00 ./deng
roc      33363  0.0  0.0   3920   368 pts/0    S    00:14   0:00 ./denga
roc      33364  0.0  0.0   3920   364 pts/0    S    00:14   0:00 ./dengb
```

试试 Ctrl+Z 组合键

假如执行 sleep 2000 时忘记加&了，则这个命令就会一直霸占着我的终端，这可怎么办？莫急莫急，有办法：

按键盘上的 Ctrl+Z 组合键，可以将前台作业切换到后台。但务必注意的是，用组

合键 Ctrl+Z 的话，这个任务到了后台会变成 Stopped 的状态。（你都无情地把它 Ctrl+Z 到后台了，它怎么可能还会任劳任怨地继续在后台工作呢？）

```
[roc@roclinux ~]$ sleep 2000
^Z
[1]+  Stopped                 sleep 2000
```

如果你仍然希望这个作业在后台处于 Running 状态，那么我们需要 bg 命令来帮忙。

Ctrl+Z 与 bg 是一家

由于组合键 Ctrl+Z 会让一个作业切换到后台后变为 Stopped 状态，而我们想让它在后台继续运行，该怎么实现呢？

用 bg 就好了，bg 是 background 的缩写，表示将一个后台处于 Stopped 状态的作业变为 Running 状态。

如果你用 jobs 命令发现有一个显示为[Stopped]的任务，想让这个任务继续在后台执行，那么就这样做：

```
#查看到了处于Stopped状态的作业
[roc@roclinux ~]$ jobs
[1]+  Stopped                 ./deng
[2]-  Running                 ./denga &

#使用bg让它在后台继续运行
[roc@roclinux ~]$ bg %1
[1]+ ./deng &

#再查看，已经在后台继续运行了
[roc@roclinux ~]$ jobs
[1]-  Running                 ./deng &
[2]+  Running                 ./denga &
```

想看看我都有哪些作业

其实在刚才的例子里，我们已经知道了如何查看当前终端的作业列表，那就是 jobs 命令：

```
[roc@roclinux ~]$ jobs
[1]   Running                 ./deng &
[2]  Running                  ./denga &
[3]-  Running                 ./dengb | ./dengd &
[4]+  Running                 sleep 2000 &
```

卖一个关子，如果你想知道[3]和[4]后面的减号（-）和加号（+）的含义，就请大

家坚持看完本篇文章啦。

fg 就好像网络推手

网络推手是把没有出名的人炒作成网红，而 fg 的作用恰恰也是把在后台的作业推到前台。

比如，我想把刚才运行的 sleep 命令放回前台来：

```
[roc@roclinux ~]$ fg %4
sleep 2000
```

fg 表示 frontground，是前台的意思，而 fg 的作用就是将一个置于后台的任务切换回前台。%4 表示的是在 jobs 命令中列出的第 4 号任务。如果是 2 号任务呢，你该知道如何写吧。

这个杀手有点冷

我想让后台的任务号是 3 的任务终止掉，也就是杀掉它，那就要派出 kill 这位杀手了：

```
[roc@roclinux ~]$ kill %3
```

用 kill %N 的方式可以杀死任务号为 *N* 的后台任务，这和找到任务的 PID 然后 kill 掉是一个效果。

说说作业的名字

在文章进入尾声之际，我们回顾全文会发现，我们一般使用“%N”这样的形式来指代一个作业（Job），但其实，我们还可以用更丰富的方式来定位到具体的作业，所以，我们来总结一下“指代作业名字”的所有方法，如表 22 所示。

表 22 指代作业名字的方法列表

符 号	含 义	示 例
%Number	Number 要求是正整数，根据编号指代某一作业	fg %1
%String	匹配命令行以 String 开头的作业，如果匹配到多个则会报错	kill %deng
%?String	命令行中含有 String 字符串的作业，若是通过管道连接的多个命令，则仅匹配第一个命令	kill %?ng

续表

符　号	含　义	示　例
%%	指代在作业列表中最近一个被切换到后台的作业	kill %%
%+	和%%作用完全相同	kill %+
%-	指排在%%所指代的作业前面的那个作业	fg %-

通过表格来说明有的不太直观，我们还是采用最接地气的方法，来让大家更直观的感受吧。

演示%String 的用法：

```
[roc@roclinux ~]$ jobs
[1]+  Running                 ./deng | ./denga | ./dengb &

[roc@roclinux ~]$ kill %./deng
[1]+  已终止                  ./deng | ./denga | ./dengb
```

演示%?String 的用法：

```
[roc@roclinux ~]$ jobs
[1]+  Running                 ./deng | ./denga | ./dengb &

[roc@roclinux ~]$ kill %?engb
-bash: kill: engb: ambiguous job spec

[roc@roclinux ~]$ kill %?eng
[1]+  已终止                  ./deng | ./denga | ./dengb
```

演示%%的用法：

```
#先启动第一个后台作业
[roc@roclinux ~]$ ./denga &
[1] 33413

#启动第二个后台作业
[roc@roclinux ~]$ ./dengb &
[2] 33414

#启动第三个后台作业
[roc@roclinux ~]$ ./dengc &
[3] 33415

#整体查看作业列表，细心的话，你会发现[2]后有减号，[3]后有加号，正好分别对应%-和%+

[roc@roclinux ~]$ jobs
[1]   Running                 ./denga &
[2]-  Running                 ./dengb &
[3]+  Running                 ./dengc &
```

```
#操作一下%1作业
[roc@roclinux ~]$ fg %1
./denga
^Z
[1]+  Stopped                 ./denga

#继续查看，减号和加号的位置变化了，可见%+指代最后一次被放于后台的作业
[roc@roclinux ~]$ jobs
[1]+  Stopped                 ./denga
[2]   Running                 ./dengb &
[3]-  Running                 ./dengc &

#使用%-符号
[roc@roclinux ~]$ kill %-
[3]-  已终止                  ./dengc

#继续查看作业列表
[roc@roclinux ~]$ jobs
[1]+  Stopped                 ./denga
[2]-  Running                 ./dengb &

#使用%+符号
[roc@roclinux ~]$ kill %+

[1]+  Stopped                 ./denga
```

21 用 trap 捕捉那神秘的信号

信号似狼烟

信号在操作系统中是一个高级货，一般人都不敢去触碰它，但从应用的角度来看，其实它还蛮简单的。下面我们就来看看有关信号的介绍。

信号是一种进程间的通信手段，它为进程提供了一种异步的软件中断机制。当进程接收到一个信号后，会有三种可能的处理方式：

- 忽略这个信号。
- 触发默认动作。
- 捕捉信号，并执行对应的处理函数。

信号是一个比较抽象的概念，有的人可能会云里雾里。所以，我们用一种比喻的手法，让大家能够更加形象地理解信号的作用。

信号，就好像古代烽火台上的狼烟，它可以给镇守城镇的主帅提供危险预警功能。当主帅看到狼烟四起时，可以有三种选择：即“无视”、“逃跑”或“抵抗”。主帅如果选择了“无视”，那就是认为什么事情都没有发生。当然主帅也可以选择“逃跑”，带着他的士兵弃城而逃。而主帅还可以选择“抵抗”，一般是根据设定好的战术打击敌人。

通过这样比较形象的比喻，大家对于信号的认识应该会更加清晰一些了吧！

捕获信号并响应

如果我们想捕获某些特定的信号，并针对这些信号采取一些特定的动作，那么我们可以借助 trap 命令来实现。

trap 命令的语法格式如下：

```
$ trap "commands" signal-list
```

首先，我们来介绍 signal-list 参数，通过这个参数可以指定我们要捕获的信号的列表。

而 commands 参数，可以指定为任何有效的 Linux 命令或用户自定义函数。

当进程捕获到 signal-list 中设定的信号后，就会自动调用 commands 中设定的动作。

百闻不如一见，我们先来做一个实验，让大家真切地感受下 trap 的威力。

我们要实现的效果是这样的：尝试捕获 INT 信号（即大家按下组合键 Ctrl+C 时所产生的信号），当捕获 INT 信号后显示出“hello world”。

```
[root@roclinux ~]# cat catchsigal.sh
#!/bin/bash

#编号2的信号，就是INT信号
trap 'echo "hello world"' 2

#我们调用tail -f，只是为了让这个脚本别退出而已
tail -f /var/log/messages
```

执行脚本后，我们按下组合键 Ctrl+C，它会触发 INT 信号：

```
[root@roclinux ~]# ./catchsigal.sh
Feb 24 11:37:41 leo NetworkManager[1107]: <info>  address 192.168.0.29
Feb 24 11:37:41 leo NetworkManager[1107]: <info>  prefix 16 (255.255.0.0)
Feb 24 11:37:41 leo NetworkManager[1107]: <info>  classless static route
169.254.169.254/32 gw 192.168.0.1
Feb 24 11:37:41 leo NetworkManager[1107]: <info>  gateway 192.168.0.1
Feb 24 11:37:41 leo NetworkManager[1107]: <info>  hostname
'host-192-168-0-29'
Feb 24 11:37:41 leo NetworkManager[1107]: <info>  nameserver
'192.168.0.3'
Feb 24 11:37:41 leo NetworkManager[1107]: <info>  nameserver
'192.168.0.13'
Feb 24 11:37:41 leo dhclient[1139]: bound to 192.168.0.29 -- renewal in
34960 seconds.
Feb 24 12:25:48 leo yum[10928]: Installed: libevent-1.4.13-4.el6.x86_64
Feb 24 12:25:48 leo yum[10928]: Installed: tmux-1.6-3.el6.x86_64
^Chello world
[root@roclinux ~]#
```

按下组合键 Ctrl+C 的一瞬间，我们看到了期盼已久的“hello world”。

其实捕获信号技术在 Shell 编程中十分常用，一般会应用到下面这些场景中：

- 动态读取并更新配置文件。
- 定期清除临时文件。
- 忽略某信号对程序可能的影响，比如组合键 Ctrl+C。

- 在用户要退出程序时，询问用户是否真的确认要退出。

屏蔽信号

上面的示例仅仅改变了原来信号的响应(不仅中断了程序，还正常输出了提示语)，但大多数时候，我们需要的是信号的屏蔽。而信号屏蔽，本质上就是忽略特定的信号。

屏蔽信号的方法如下：

```
$ trap "" signal-list
```

下面我们就动手来屏蔽一下 INT 信号吧：

```
[root@roclinux ~]# trap "" INT
```

请注意这里，和上面不同之处是 commands 部分被设置成了“”，也就是捕捉到信号后不做任何动作，这样就实现了信号的屏蔽，我们来实际试一试。

我们尝试执行一条不会退出的命令，然后按下组合键 Ctrl+C。

```
[root@roclinux ~]# tail -f /var/log/messages
```

我们连续多次按下 Ctrl+C 组合键，程序纹丝不动，果然没有退出，看来我们的信号屏蔽真的起到了作用。

可是，在这种情况下，我们该如何退出 tail 程序呢？试试组合键 Ctrl+\吧，或许会管用哦！

恢复信号

如果我们想把某些信号的处理动作恢复到默认情况，也很简单，那就是：

```
$ trap signal-list
```

原来，只要我们直接设置信号，而不设置 commands 部分，就可以恢复信号的处理动作了。

```
#赶快把INT信号的恢复回来
[root@roclinux ~]# trap INT

# 继续尝试执行tail
[root@roclinux ~]# tail -f /var/log/messages
Apr    7  05:04:10  leo  dhclient[1138]:  DHCPACK  from  192.168.0.13
(xid=0x3b14b50c)
```

```
Apr  7 05:04:10 leo NetworkManager[1109]: <info> (eth0): DHCPv4 state
changed renew -> renew
Apr  7 05:04:10 leo NetworkManager[1109]: <info>  address 192.168.0.29
Apr    7  05:04:10  leo  NetworkManager[1109]:  <info>     prefix  16
(255.255.0.0)
Apr  7 05:04:10 leo NetworkManager[1109]: <info>  classless static route
169.254.169.254/32 gw 192.168.0.1
Apr  7 05:04:10 leo NetworkManager[1109]: <info>  gateway 192.168.0.1
Apr    7  05:04:10  leo  NetworkManager[1109]:  <info>     hostname
'host-192-168-0-29'
Apr    7  05:04:10  leo  NetworkManager[1109]:  <info>     nameserver
'192.168.0.3'
Apr    7  05:04:10  leo  NetworkManager[1109]:  <info>     nameserver
'192.168.0.13'
Apr  7 05:04:10 leo dhclient[1138]: bound to 192.168.0.29 -- renewal in
34064 seconds.
^C
# 按组合键Ctrl+C 执行，程序正常终止
[root@roclinux ~]#
```

程序正常退出了，这也说明我们的信号处理恢复到了默认的情况了。

篇尾彩蛋——常见的信号

常见的信号如表 23 所示。

表 23 常见的信号

信号名称	信号数	描　　述
SIGHUP	1	本信号在用户终端连接（正常或非正常）结束时发出，通常是在终端的控制进程结束时，通知同一 Session 内的各个作业，这时它们与控制终端不再关联。登录 Linux 时，系统会分配给登录用户一个终端（Session）。在这个终端运行的所有程序，包括前台进程组和后台进程组，一般都属于这个 Session。当用户退出 Linux 登录时，前台进程组和后台有对终端输出的进程将会收到 SIGHUP 信号。这个信号的默认操作为终止进程，因此前台进程组和后台有终端输出的进程就会中止。对于与终端脱离关系的守护进程，则这个信号用于通知它重新读取配置文件
SIGINT	2	程序终止（interrupt）信号，在用户键入 INTR 字符（通常是组合键 Ctrl+C）时发出
SIGQUIT	3	和 SIGINT 类似，但由 QUIT 字符（通常是组合键 Ctrl+\）来控制。进程在因收到 SIGQUIT 退出时会产生 core 文件，在这个意义上类似于一个程序错误信号
SIGFPE	8	在发生致命的算术运算错误时发出。不仅包括浮点运算错误，还包括溢出及除数为 0 等其他所有的算术的错误

续表

信号名称	信号数	描　　述
SIGKILL	9	用来立即结束程序的运行。本信号不能被阻塞、处理和忽略。
SIGALRM	14	时钟定时信号，计算的是实际的时间或时钟时间. alarm 函数使用该信号
SIGTERM	15	程序结束（terminate）信号，与 SIGKILL 不同的是，该信号可以被阻塞和处理。通常用来要求程序自己正常退出。Shell 命令 kill 缺省产生这个信号。

如果你想学习一下 Linux 中的所有信号，则可以通过下面的方式找到所有的信号。

```
[root@roclinux ~]# trap -l
 1) SIGHUP    2) SIGINT    3) SIGQUIT   4) SIGILL    5) SIGTRAP
 6) SIGABRT   7) SIGBUS    8) SIGFPE    9) SIGKILL  10) SIGUSR1
11) SIGSEGV 12) SIGUSR2 13) SIGPIPE 14) SIGALRM 15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD 18) SIGCONT 19) SIGSTOP 20) SIGTSTP
21) SIGTTIN 22) SIGTTOU 23) SIGURG  24) SIGXCPU 25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF 28) SIGWINCH 29) SIGIO   30) SIGPWR
31) SIGSYS  34) SIGRTMIN 35) SIGRTMIN+1  36) SIGRTMIN+2  37)
SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42)
SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47)
SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52)
SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57)
SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62)
SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

好了，信号有很多，但常用的不多，信号的知识不仅包括本文的内容，还有很多 Linux 操作系统级的信号知识，建议大家阅读《UNIX 环境高级编程》一书来深造！

22 nohup，强大的防弹护甲

给你一副 nohup 护甲

终端是我们进入 Linux 系统的入口，无论是上线操作、排查问题，还是试验学习，面对的都是 Linux 终端。

当用户注销、网络断开时，终端就会收到 SIGHUP（hangup）信号，然后关闭其所有从属的子进程。如果我们不希望某些进程被这个 SIGHUP 信号所影响，就需要为这些进程配备上 nohup 护甲。

顾名思义，nohup 就是“say no to hangup”，有了 nohup 这层防弹护甲，就可以对 SIGHUP 信号免疫啦。

有人说，直接在命令后面加入后台符（&）就能将命令放到后台啦，还需要 nohup 作什么呢？其实这是很多人的误解，使用后台符（&）只是让进程躲到当前终端的后台去了，SIGHUP 仍然会波及到这些躲在后台的进程，所以需要 nohup 来彻底地防御 SIGHUP 信号，真正让进程遁地无形。

如何优雅地执行 nohup

在所要执行的命令前面加上 nohup，便可以忽略所有挂断（SIGHUP）信号，铜头铁臂的感觉是不是很爽？如果在命令后再加上&（表示“and”的符号），那就可以让命令在穿上防弹护甲的同时，也拥有遁地无形的超能力，命令将直接变成后台作业，你可以继续在当前终端执行其他工作，由此可以看出，nohup 和&简直就是天生的一对。

下面是 nohup 的命令格式。

```
nohup 具体要执行的命令 [ & ]
```

如果不将 nohup 命令的输出重定向，则输出将追加到当前目录的 nohup.out 文件中。如果当前目录的 nohup.out 文件不可写（比如权限不够），输出将重定向到

$HOME/nohup.out 文件中。

```
[root@roclinux ~]# nohup ping www.baidu.com &
[1] 16263
[root@roclinux ~]# nohup: ignoring input and appending output to
‘nohup.out’
```

在上面的例子中，将 ping 命令设为后台作业去执行，返回的[1]表示该后台作业的 id 是 1，16263 代表其 PID。由于没有指定输出的重定向文件，因此默认将标准输出（STDOUT）和标准错误（STDERR）输出都追加到 nohup.out 文件的末尾。

我们可以使用 jobs 来查看 Job ID 对应的具体命令。

```
[root@roclinux ~]# jobs
[1]+  Running                 nohup ping www.baidu.com &
```

也可以使用 fg（就是 foreground 的意思）加上 Job ID，将后台命令调至前台运行。

```
[root@roclinux ~]# fg %1
nohup ping www.baidu.com
```

如果我们运行多个 nohup 命令，且没有为每个命令指定输出重定向文件，那么多个后台命令的输出都会追加到 nohup.txt 文件，非常不利于查找结果和调试程序。一个好的习惯是为每一个 nohup 指定不同的输出文件。

```
[root@roclinux ~]# nohup ping www.baidu.com > ping_baidu.txt&
[root@roclinux ~]# nohup: ignoring input and redirecting stderr to stdout
```

上面的命令虽然给标准输出指定了重定向文件 ping_baidu.txt，但是并没有显示为标准错误指定重定向文件，因此会有提示信息告知标准错误将重定向到标准输出中去，也就是 ping_baidu.txt 文件。

看到这些提示信息会不会觉得很烦，我们可以使用下面的命令来跟这些提示信息彻底说拜拜。

```
[root@roclinux ~]# nohup ping www.baidu.com > ping_baidu.txt 2>&1 &
```

上面命令的“> output.txt”代表将标准输出（1）重定向到 output.txt，“2>&1”表示将标准错误（2）重定向到标准输出（1），所以标准错误和标准输出都导入文件 output.txt 里面去，效果相当于执行了如下命令。

```
 [root@roclinux ~]# nohup ping www.baidu.com 2> /dev/null 1>&2 &
```

如果你的命令会产生大量的输出，而且这些输出你根本不在乎，那就交给/dev/null 这个无底洞吧，它的胃口很大，可以吃掉你所有的输出。

我依然关心着后台的你

并不是把命令放到后台执行，就表示我们不关心它了，它依然是我们的心头肉。当我们想获取 nohup 命令的返回值时，可以用$!获取分配的子进程 ID，然后用 wait PID 加 echo $?的方式获取这个子进程的返回值。

```
[root@roclinux ~]# nohup rm /tmp/nonexist.file > nohup.out  2>&1 &
[1] 16333

[root@roclinux ~]# PID=$!
[1]+  Exit 1                  nohup rm /tmp/nonexist.file > nohup.out 2>&1

[root@roclinux ~]#  echo $PID
16332

[root@roclinux ~]# wait $PID

[root@roclinux ~]# echo $?
1
```

上面的例子中，我们删除了一个不存在的文件/tmp/nonexist.file，并将删除命令放于后台执行。然后获取到对应的 PID 之后，使用 wait 的方式来等待 nohup 执行完毕，然后赶紧将返回值 echo 出来。可以看出，这里 nohup 的返回值 1 就是 rm 命令的返回值。

在大多数情况下，nohup 的返回值就是其对应命令的返回值，但是也有例外：

- 125：如果 nohup 命令失败，并且 POSIXLY_CORRECT 环境变量没有设置。
- 126：指定命令能找到，但是不能调用。
- 127：找不到指定命令。

nohup 的同门师兄弟

nohup 是唯一一个防弹护甲吗？当然不是，它的同门师兄弟还有 screen、setsid 和 disown。不过正如华山派还分剑宗和气宗一样，Linux 的硬气功界也分为两个门派，要想让运行的命令不因用户注销、网络断开等因素而中断，有两个基本思路：

- 让进程对 SIGHUP 信号免疫。免疫宗的成员有 nohup 和 disown。
- 让进程在新的会话中运行。会话宗的成员有 setid 和 screen。

接下来让我们一一拜会一下。

免疫宗 disown

世上有后悔药吗？现实生活中没有，Linux 世界里可以有。如果在运行命令之前忘记加上 nohup，而命令运行到一半，你是不是在犹豫要不要 kill 掉进程重新来过？其实大可不必，通过作业调度和 disown 就能够解决这个问题了。

首先，将当前正在前台运行的进程放到后台运行通过（Ctrl+Z 组合键和 bg 命令实现），然后执行“disown -h %{jobid}”，这里的{jobid}是通过 jobs 命令中看到的进程前[]中的数字。

下面的例子中，我们一开始运行 ping，将输出重定向到 output.txt 中。随后执行 Ctrl+Z 组合键将当前进程挂起，可以看到作业号为 1。

```
[root@roclinux ~]# ping www.baidu.com > output.txt
^Z
[1]+  Stopped                 ping www.baidu.com > output.txt
```

然后，执行 bg 将该作业放入后台，可以看到 ping 命令后面出现了一个&符号。通过执行 jobs 命令，我们可以看到该作业正在后台 running。

```
[root@roclinux ~]# bg %1
[1]+ ping www.baidu.com > output.txt &

[root@roclinux ~]# jobs
[1]+  Running                 ping www.baidu.com > output.txt &
```

最后，执行 disown 命令，它会将指定作业从作业列表中移除，我们将不能再使用 jobs 来查看它，但是我们依然能够用 ps -ef 查找到它。

```
 [root@roclinux ~]# disown -h %1

[root@roclinux ~]# ps -ef |grep ping
root     22875 22778  0 03:10 pts/0    00:00:00 ping www.baidu.com
root     22877 22778  0 03:11 pts/0    00:00:00 grep --color=auto ping
```

通过这种方式，我们就实现了类似 nohup 的效果。

```
  00:00:00 grep --color=auto ping
```

会话宗 setid

setsid 命令，是（set session id 的缩写，它）能够让进程在一个新的会话中运行，从而避开当前会话的 SIGHUP 信号。我们只需在要执行处理的命令前加上 setsid 即可。

```
[root@roclinux ~]# setsid ping www.ibm.com
[root@roclinux ~]# ps -ef |grep www.ibm.com
root     31094     1  0 07:28 ?       00:00:00 ping www.ibm.com
root     31102 29217  0 07:29 pts/4    00:00:00 grep www.ibm.com
```

会话宗 screen

nohup 和 disown 命令可以让进程免受 SIGHUP 信号的影响，但是如果有大量的命令需要在后台运行，如何避免对每条命令都做这样的操作呢？当当当，同门大师兄 screen 闪亮登场，它能够在一个真实终端下运行多个全屏的伪终端，可以认为是开启了多个会话。这种方法同样可以避开 SIGHUP 信号的猎杀。

下面的例子中，我们通过 screen 命令开启了一个叫作 screen_ping 的伪终端，并在其中运行 ping 命令。

```
[root@roclinux ~]#screen -A -m -d -S screen_ping ping www.baidu.com  &
```

使用 screen -list 可以列出所有运行中的会话。

```
[root@roclinux ~]# screen -list
There is a screen on:
       23231.screen_ping      (01/22/2016 03:24:46 AM)       (Detached)
1 Socket in /var/run/screen/S-root.
```

系统管理篇

在系统管理篇中，我们将为大家带来 9 篇文章，所有内容都是围绕着 Linux 系统管理展开的，包括：

在这里，你不仅可以看到大片“我是谁”，可以感受到 Linux 中森严的等级制度，还可以了解 shutdown、poweroff、reboot 和 halt 的区别之处。

让我们现在就进入系统管理篇的学习之旅吧！

1 uname 展示系统信息

uname 一个用法走天下

uname 的用法很好记，只要记住-a 选项，基本就够了，我们来看一下：

```
[roc@roclinux ~]$ uname -a
Linux roclinux 2.6.32-220.4.1.el6.x86_64 #1 SMP Tue Jan 24 02:13:44 GMT
2012 x86_64 x86_64 x86_64 GNU/Linux
```

这里的一大长串输出，包括了我们系统中的若干个重要信息，我们通过一张图来展示，如图 20 所示。

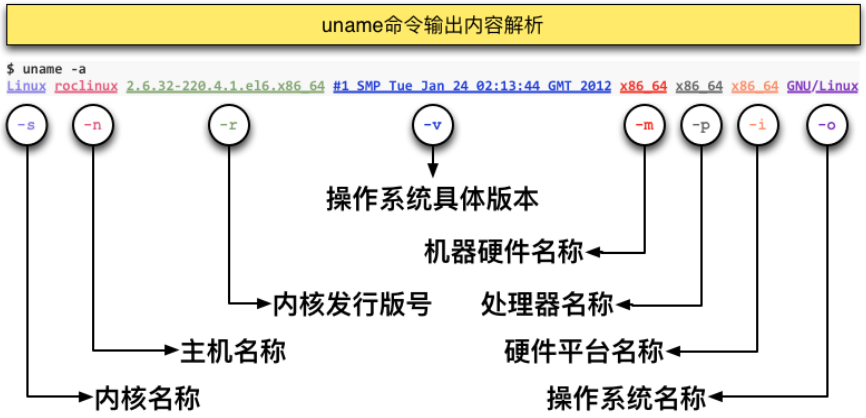


图 20　uname 命令输出内容解析

uname 的选项绝对是屈指可数，我们可以很快地展示一遍：

```
#内核名称
[roc@roclinux ruanjian]$ uname -s
Linux

#主机名称
[roc@roclinux ruanjian]$ uname -n
roclinux
```

```
#内核发行版号
[roc@roclinux ruanjian]$ uname -r
2.6.32-220.4.1.el6.x86_64

#操作系统具体版本
[roc@roclinux ruanjian]$ uname -v
#1 SMP Tue Jan 24 02:13:44 GMT 2012

#机器硬件名称
[roc@roclinux ruanjian]$ uname -m
x86_64

#处理器名称
[roc@roclinux ruanjian]$ uname -p
x86_64

#硬件平台名称
[roc@roclinux ruanjian]$ uname -i
x86_64

#操作系统名称
[roc@roclinux ruanjian]$ uname -o
GNU/Linux
```

好了，有关 uname 的用法全部都在这里了。相信聪明的同学们可以很快记住的。

2 用户 ID 和用户组 ID 的一些故事

实际用户和有效用户

实际用户，英文术语叫作 real user id，就是登录 Shell 的那个时刻所使用的用户 ID。用 who am i 所展示出来的就是“实际用户”。

登录进 Shell 之后，使用 su 或 su -切换到的用户 ID，可就不叫实际用户了，需要有另一个名词来指代。

所以，linux 建立了另一个概念，叫作有效用户，英文术语叫作 effective user id，它代表的便是我们用 su 或者 su -所切换到的那个用户。用 whoami 所展示出来的就是“有效用户”。

这个时候，如果我们执行了一个命令，这个命令产生了一个进程，那么，这个进程所属的 User 就是有效用户 ID，我们来看一个实际例子：

```
#我们当前的有效用户是roc
[roc@roclinux ~]$ whoami
roc

#我们运行一个程序，它会产生一个进程
[roc@roclinux ~]$ ./deng &
[1] 21619

#我们查看此进程，第一列展示的就是有效用户，看，正是roc
[roc@roclinux ~]$ ps auxww|grep deng
roc      21619  0.0  0.0   3920   368 pts/6    S    11:55   0:00 ./deng
```

我们切换到 root 有效用户，再来看一遍：

```
#切换到root
[roc@roclinux ~]$ sudo su

#看，有效用户从roc变成了root
[root@roclinux ~]# whoami
root
```

```
#我们依旧运行一个命令，产生一个进程
[root@roclinux ~]# ./deng &
[1] 21784

#看，这个进程的所属用户变成了root
[root@roclinux ~]# ps auxww|grep deng
root     21784  0.0  0.0   3920   368 pts/6    S    11:57   0:00 ./deng
```

用户组的种种

用户组，是 Linux 初学者往往会忽视的一个概念。不少新手都不太会通过用户组管理用户及其权限，甚至对用户组的各种概念也会漠然无视。

但通过实践和经验来看，对用户组的巧妙使用是可以事半功倍的，而且是高级 Linux 管理员必备的技能之一。

在接下来的这一段落中，我们就来介绍用户组的一些重要概念。

第一个是“初始用户组”。

还记得/etc/passwd 文件里的第 4 栏么（忘了的话，可以用 cat /etc/passwd 去看下），它表示 GID，就是“用户组 ID”，我们称这个值为“初始用户组”。当用户登录系统时，立刻就拥有了这个用户组的相应权限：

```
[root@roclinux ~]# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
(此处省略数百行)
```

如果你曾经研究过用户组的相关知识，那么应该会知道/etc/group 文件，这个文件里面储存着所有的用户组名称以及相应的用户账号列表。

值得注意的一点是，一般情况下这个用户账号列表里是不存储“初始用户组名称”的，因为初始用户组名称在/etc/passwd 里已经存储了，没必要在这里再重复存储了。

这句话稍显晦涩。举个例子描述就是，假如 userA 用户属于 groupA 用户组，且 groupA 是 userA 的初始用户组，那么在/etc/group 的 groupA 一行里，其实不必再列出 userA 啦，只要列出 groupA 所包含的其他用户就可以了。因为在/etc/passwd 中已经记录了 userA 和 groupA 的包含关系了。

我们来看一下/etc/group 这个文件：

```
#格式：用户组名称:加密标识:用户组ID:所包含的用户账号
[root@roclinux ~]# cat /etc/group
root:x:0:root
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon
sys:x:3:root,bin,adm
adm:x:4:root,adm,daemon
tty:x:5:
disk:x:6:root
lp:x:7:daemon,lp
mem:x:8:
kmem:x:9:
(此处省略数十行)
```

了解了初始用户组之后，我们还需要了解另一个重要概念，即“有效用户组”。

有效用户组表示用户此时此刻所在的用户组是什么。有些同学会问“难道用户所在的用户组还会变来变去么？”

答案是肯定的。这都是 newgrp 命令搞的鬼，它可以帮助用户转换到不同的用户组，比如 newgrp group123，就可以让用户转换到 group123 用户组。

请注意，这种转换的背后原理是“创建一个新的子 Shell”，而如果你想恢复到原来用户组的话，方法也很简单，那就是用 exit 或 Ctrl+D 组合键来退出这个 Shell 即可。

正如你所看到的，有效用户组其实就是用 newgrp 命令所切换到的用户组。当然，如果你一次也没有使用过 newgrp 命令，那么有效用户组就是初始用户组。

另外，你可以用 groups 命令来列出当前用户所支持的所有用户组。（在有些系统中，第一个列出的就是有效用户组，但有些系统可能并非如此）

```
#root用户属于好几个用户组
[root@roclinux ~]# groups
root bin daemon sys adm disk wheel

#而roc用户只属于同名的roc用户组
[roc@roclinux ~]$ groups
roc
```

还有一点应该注意，就是当你新建了一个文件时，此文件的所属用户组就是创建人当前的有效用户组哦。

3 whoami 不只是一部电影

who 命令里暗藏玄机

今天我们要说的“Who Am I”不是成龙在 1998 年拍摄的香港动作大片《我是谁》，而是 Linux 系统中的 who 命令。

有些同学可能会问了：这个命令好奇怪，哪有问“我是谁”的，账号是你自己输入的，密码也是你自己输入的，怎么可能不知道自己是谁呢？

哈哈，你还别说，这里面玄机重重，你还真未必知道自己是谁。

我们今天就来看看这个“Who Am I”里面的玄机。

玄机一——whoami 和 who am i

Who 命令中，最常见的用法有三种，它们是：

- who
- whoami
- who am i

我们先来看看这三种用法的输出信息有什么不同：

```
[roc@roclinux ~]$ whoami
roc

[roc@roclinux ~]$ who am i
roc      pts/4        2016-02-29 10:08 (221.219.109.12)

[roc@roclinux ~]$ who
roc      pts/0        2016-02-29 09:49 (221.219.109.12)
roc      pts/4        2016-02-29 10:08 (221.219.109.12)
yangwenqiang pts/5        2016-02-29 10:09 (221.219.109.12)
```

当我用 sudo su（或者 sudo su -）切换到 root 用户之后，再来看看输出的变化：

```
[roc@roclinux ~]$ sudo su

[root@roclinux roc]# whoami
root

[root@roclinux roc]# who am i
roc      pts/0        2016-02-29 09:49 (221.219.109.12)

[root@roclinux roc]# who
roc      pts/0        2016-02-29 09:49 (221.219.109.12)
roc      pts/4        2016-02-29 10:08 (221.219.109.12)
yangwenqiang pts/5        2016-02-29 10:09 (221.219.109.12)
```

看出区别了么，区别其实就在 whoami 这个命令。

我们用 Linux 正规描述，再结合“面具”的类比，来为大家解释这第一个玄机，如表 24 所示。

表 24　命令对比

命　令	对应账户	含　义	类　比
whoami	root	显示的是当前操作用户的用户 ID，术语叫作“有效用户 ID”	面具
who am i	roc	显示的是当初登录进系统时的用户 ID，术语叫作“实际用户 ID”。	面具背后的真实面孔

好了，第一个玄机已经真相大白了，我们继续来说说下一个玄机。

玄机二——who am i 和 who are you

这个玄机，其实很好理解，我们用例子来展示就好啦：

```
#正规写法
[root@roclinux ~]# who am i
roc      pts/4        2016-02-29 10:08 (221.219.109.12)

#这也行……
[root@roclinux ~]# who are you
roc      pts/4        2016-02-29 10:08 (221.219.109.12)

#这也可以……
[root@roclinux ~]# who is she
roc      pts/4        2016-02-29 10:08 (221.219.109.12)

#这还行……
[root@roclinux ~]# who xxx yyy
roc      pts/4        2016-02-29 10:08 (221.219.109.12)

#开始数数了……
[root@roclinux ~]# who 1 2
roc      pts/4        2016-02-29 10:08 (221.219.109.12)

#终于不行了……
[root@roclinux ~]# who 1 2 3
who：额外的操作数 "3"
请尝试执行"who --help"来获取更多信息。
```

哈哈，看明白了么，我们已经揭开了玄机二的谜底，那就是 who 其实并不在乎参数一和参数二的内容，它只在乎是否存在参数一和参数二，一旦存在，其就会展示实际用户 ID。

当然，who 命令还是有底线的，如果参数超过两个，它就不支持了。

玄机三——w 和 who

两者的作用很是相像，而且有一部分是重叠的，这从两者的 man 介绍中，也可见一斑：

```
who - show who is logged on
w   - Show who is logged on and what they are doing.
```

如你所见，w 除了展示都有哪些用户登录之外，还展示了它们分别正在做什么，区别仅此而已。

我们输入 w，得到的信息是这样的：

```
[root@roclinux ~]# w
 11:11:15 up 109 days, 18:25,  3 users,  load average: 0.09, 0.17, 0.14
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
roc      pts/0    221.219.109.12   09:49   21.00s  0.33s  0.00s /bin/bash
/home/roc/bin/t
roc      pts/4    221.219.109.12   10:08    0.00s  0.62s  0.03s sshd: roc
[priv]
yangwenq pts/5    221.219.109.12   10:09    1:01m  0.27s  0.27s -bash
```

我们输入 who，得到的信息是这样的：

```
[root@roclinux ~]# who
roc      pts/0        2016-02-29 09:49 (221.219.109.12)
roc      pts/4        2016-02-29 10:08 (221.219.109.12)
yangwenqiang pts/5        2016-02-29 10:09 (221.219.109.12)
```

玄机四——roc 和 root

有些同学又发现问题了：为什么我 sudo su -到 root 之后，who 里面却没显示出 root 呢？

比如下面的例子，我已经 sudo su -到 root 了，但是所展示的仍然只有 roc：

```
[root@roclinux ~]# who
roc      pts/0        2016-02-29 09:49 (221.219.109.12)
roc      pts/4        2016-02-29 10:08 (221.219.109.12)
yangwenqiang pts/5        2016-02-29 10:09 (221.219.109.12)
```

能问出这个问题，说明你观察得很仔细，而且也善于发现问题。

这是因为 su 切换的用户进程空间是作为一个子空间存在的，他并没有得到一个登录的 tty 终端，而 who 是按照不同 tty 来展示登录信息的。

你应该知道的 who 的其他选项

有时候，你会见到 who -m 这样的用法，其实它完全等价于 who am i，也是展示实际用户 ID。

```
[root@roclinux ~]# who am i
roc      pts/4        2016-02-29 10:08 (221.219.109.12)

# 用-m选项
[root@roclinux ~]# who -m
roc      pts/4        2016-02-29 10:08 (221.219.109.12)
```

如果你只想展示出当前登录的用户个数，那么请用-q 选项，它会列出当前登录的用户个数以及具体的名单列表：

```
[root@roclinux ~]# who -q
roc roc yangwenqiang
# 用户数=3
```

可以看出，虽然是同一个账户名（roc），但如果登录到不同的 tty 终端，也会认为是不同的用户。

当你觉得 who 的输出信息晦涩难懂时，可以使用 who -H 来输出，这样可以在每列上面加上列名称，有助于阅读：

```
[roc@roclinux ~]$ who -H
名称     线路         时间            备注
roc      pts/0        2016-02-29 09:49 (221.219.109.12)
roc      pts/4        2016-02-29 10:08 (221.219.109.12)
yangwenqiang pts/5        2016-02-29 10:09 (221.219.109.12)
```

另外，我们用一组示例再来看看 who 还能给我们哪些惊喜：

```
# 使用who的-b选项，可以列出系统上一次的启动时间
[roc@roclinux ~]$ who -b
         系统引导     2015-11-11 16:39

# 使用who的-r选项，可以列出系统当前的运行级别
[roc@roclinux ~]$ who -r
        运行级别 3   2015-11-11 16:39

# 使用who的-l选项（--login），可以列出"系统登录进程"的信息
[roc@roclinux ~]$ who -l
登录     tty1         2015-11-11 16:39             1268 id=1
登录     tty3         2015-11-11 16:39             1272 id=3
登录     tty2         2015-11-11 16:39             1270 id=2
登录     tty4         2015-11-11 16:39             1274 id=4
登录     tty5         2015-11-11 16:39             1277 id=5
登录     tty6         2015-11-11 16:39             1281 id=6

# 如果你使用-a选项，则可以列出who能列出的几乎所有信息
[roc@roclinux ~]$ who -a
          系统引导     2015-11-11 16:39
          运行级别 3   2015-11-11 16:39
登录      tty1         2015-11-11 16:39             1268 id=1
登录      tty3         2015-11-11 16:39             1272 id=3
登录      tty2         2015-11-11 16:39             1270 id=2
登录      tty4         2015-11-11 16:39             1274 id=4
登录      tty5         2015-11-11 16:39             1277 id=5
登录      tty6         2015-11-11 16:39             1281 id=6
roc      + pts/0        2016-02-29 09:49   .         18848 (221.219.109.12)
          pts/1        2016-02-03 00:29              560 id=ts/1  终端=0 退
出=0
```

```
          pts/2       2016-02-18 23:10           52731 id=ts/2
终端=0 退出=0
          pts/3       2016-01-26 11:59           26035 id=ts/3
终端=0 退出=0
roc      + pts/4       2016-02-29 10:08   .       19109 (221.219.109.12)
          pts/9       2016-01-23 14:35            3819 id=ts/9
终端=0 退出=0
yangwenqiang + pts/5       2016-02-29 10:09 01:12      19207
(221.219.109.12)
```

惊喜连连，是不是对 who 有了全新的认识，赶快和你周围的小伙伴宣传宣传吧！

4 service 服务最周到

service 何许人也

service 是一个用来“运行 System V 初始化脚本”的工具，官方英文原文是这样表达的：

```
service - run a System V init script
```

有些同学会问，什么是 System V 呢？借此机会，我们先和大家科普一下这个名词。

System V 何许人也

System V，是历史上很著名的一个 UNIX 操作系统的分支版本，它的全称叫作 AT&T System V，可见它是由 AT&T 来主导开发的。

System V 总共只发布过 4 个正式版本，反响最好最成功的当属 System V Release 4 这一版本，大家都亲切的称之为“SVR4”。

SVR4 的很多设计，后来都成为了 UNIX 家族中很多操作系统的参照源头和事实上的标准，比如，我们今天文章的主角“System V 初始化脚本”，就是其中之一。

听说 SVR4 的地位和影响如此的重要，一定有不少同学想去下载和研究 SVR4 的源码和思想了。但这并不太容易，因为 System V 是受版权严格控制的，而非开源的系统。如果你真的感兴趣，建议你下载 Solaris 的源码就可以了，因为 Solaris 就是从 System V 发展而来的，它仍然保持着 System V 的比较纯正的血统。

service 使用初体验

聊了不少历史，我们还是抓紧时间学习一下 service 的用法吧。

假如你想禁止外部用户 ssh 远程登录本机，那么你可以执行下面的命令（要切换到 root 账户）：

```
[root@roclinux init.d]$ service sshd stop
关闭 sshd:                                                 [确定]
```

只需要这样一条简单的命令，本机的 sshd 服务就会被关闭，外部用户也就无法再通过 ssh 登录本机了。

如果我想看看本机的 ntp 服务（网络时间协议）现在是开启状态还是关闭状态呢，可以这样执行：

```
[root@roclinux init.d]# service ntpd status
ntpd 已停
```

可以看出 ntp 服务当前是关闭状态，要想开启这项服务，可以这样执行：

```
[root@roclinux init.d]# service ntpd start
正在启动 ntpd:                                             [确定]
```

如果我调整了一些 ntp 服务的参数，想要通过重启的方式使这些新配置生效，则可以这样做：

```
[root@roclinux init.d]# service ntpd restart
关闭 ntpd:                                                 [确定]
正在启动 ntpd:                                             [确定]
```

例子介绍到这里，想必你已经掌握了 service 命令的最主要功能了，其实很简单，用语法表达就是：

```
# service [服务名] [start/stop/restart/status]
```

比起复杂的 sort、find、awk 等命令，service 命令已经算是相当易学啦。

我有哪些服务名呢

现在我们已经学会了如何 start、stop、restart，但如果没有可以让我们控制的服务，那也是无济于事啊。

本节就来告诉大家我们都可以掌控哪些服务。方法很简单，如下所示：

```
[roc@roclinux init.d]$ cd /etc/init.d/
[roc@roclinux init.d]$ ls -F
auditd*          httpd*              matahari-broker*      messagebus*
nginxd@    qpidd*       saslauthd*  udev-post*
avahi-daemon*    hv_kvp_daemon*      matahari-host*            mysqld@
ntpd*      rdisc*       single*     wdapache@
crond*         ip6tables*       matahari-network*    netcf-transaction*
ntpdate*   restorecond*  snmpd*
functions        iptables*           matahari-service*     netconsole*
```

```
php-fpm@   rrdcached*    snmptrapd*
halt*               killall*             matahari-sysconfig*   netfs*
postfix*   rsyslog*      sshd*
htcacheclean*    libvirt-guests*    memcached@                  network*
pureftpd@  sandbox*      sysstat*
```

在/etc/init.d 目录下的这些文件，全部都是你可以通过 service 命令掌控的服务。它们有些是以独立的可执行文件的形式存在的，而有些则是以符号链接形式外链到一个真实的可执行文件的。

我们随意挑选一个吧，就选 rsyslog 啦，它是远程系统日志服务，我们来尝试一下开启、重启、关闭、状态这四个操作：

```
[root@roclinux init.d]# service rsyslog start
启动系统日志记录器：                                       [确定]

[root@roclinux init.d]# service rsyslog restart
关闭系统日志记录器：                                       [确定]
启动系统日志记录器：                                       [确定]

[root@roclinux init.d]# service rsyslog stop
关闭系统日志记录器：                                       [确定]

[root@roclinux ~]# service rsyslog status
rsyslogd 已停
```

service 其实只是一个脚本

service 真的只是一个脚本，你是不是会有些不相信呢？我们来揭开它的面纱：

```
#找到service工具的路径
[root@roclinux init.d]# which service
/sbin/service

#原来service真的是一个脚本
[root@roclinux init.d]# file /sbin/service
/sbin/service: POSIX Shell script text executable

#service脚本才仅仅66行，你有兴趣的话可以仔细读一下代码
[root@roclinux init.d]# wc -l /sbin/service
66 /sbin/service
```

如果你仔细阅读了脚本，你会发现其实 service 命令只是一个代理商，它本身并不能直接操控各类服务，它所做的只是把用户的操控动作（start/stop/restart/status）传递给/etc/init.d 中相应的命令而已。

所以呢，我们完全可以做如下的转换：

```
$ service ntpd stop
```

替换为

```
$ /etc/init.d/ntpd stop
```

该不该用 service 命令

“我学习笔记都做了，你现在才讨论该不该用这个命令？!”

额，不要着急，其实并没有那么严重，相信你学的这些知识在 99%的场景下都是可用的。但有些事情，是必须丑话说在前面的。

service 命令，在 RedHat 等主流 Linux 发行版中是可以用的。但有一部分 Linux 发行版中或许真的没有这个命令，所以，这个时候就要灵活处理，将 service 命令形式替换成/etc/init.d 形式。这样，你就可以走遍天下都不怕了。

5 chkconfig 掌控等级制度

森严的等级制度

说“等级制度”有些危言耸听，准确地说，应该是运行等级，英文叫作 Runlevel。

我们在学习 chkconfig 命令前，十分有必要对这个概念做一个充分的了解。那么什么是运行等级呢？

这个概念最早出现在 System V 系统的设计中，用来指定 Linux 的功能级别，不同的级别会对应着不同的功能集合。

不同的 UNIX/Linux 分支版本，都有着自己的级别制度，也分别对应着不同的功能集合，比如鼻祖 SVR4 定义了 0/1/2/3/4/5/6/s/S 等级，而 AIX 只定义了 0/1/2 三个等级而已。

本文则主要介绍 Linux 的标准等级方案，即 0/1/2/3/4/5/6 七大运行等级。

Linux 的标准运行等级

著名的 Linux 标准规范（LSB：Linux Standard Base），对运行等级有着具体的描述和规定，如表 25 所示。

表 25　Linux 标准规范之运行等级

标识数字	等级名称	描　　述
0	停机	用于关机
1	单用户模式	用于管理员系统维护
2	无网络多用户模式	禁用网络相关功能
3	有网络多用户模式	正规的系统模式
4	未使用，预留模式	一般不用
5	图形化模式	在等级 3 的基础上增加图形化展示
6	重启	用于重启系统

而每一种运行等级，都会对应着不同的服务开关方案，比如 netfs 服务在等级 2 时就被设置为关闭状态，而在等级 3 时被设置为开启状态。

隆重介绍 chkconfig

有了前面的背景介绍，我们就好解释 chkconfig 的作用了。chkconfig 是一个服务等级的管理工具，工程师可以通过这个工具来查看、调整每个运行等级所对应的服务的开关状态。

举个例子的话就是，如果我想在运行等级 2/3/4/5 上开启 netfs 服务，那么就可以用 chkconfig 来轻松实现：

```
#先查看netfs
[root@roclinux etc]# chkconfig --list netfs
netfs          0:关闭  1:关闭  2:关闭  3:启用  4:启用  5:启用  6:关闭

#开始进行设置
[root@roclinux etc]# chkconfig --level 2345 netfs on

#检查一下，果然等级2/3/4/5是启用状态啦
[root@roclinux etc]# chkconfig --list netfs
netfs          0:关闭  1:关闭  2:启用  3:启用  4:启用  5:启用  6:关闭
```

chkconfig 的几个常见用法

第 1 个用法，列出系统中各个等级对应的服务开关情况：

```
[root@roclinux ~]# chkconfig --list
auditd         0:关闭  1:关闭  2:启用  3:启用  4:启用  5:启用  6:关闭
avahi-daemon    0:关闭  1:关闭  2:关闭  3:启用  4:启用  5:启用  6:关闭
crond          0:关闭  1:关闭  2:启用  3:启用  4:启用  5:启用  6:关闭
htcacheclean    0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
httpd          0:关闭  1:关闭  2:关闭  3:启用  4:关闭  5:启用  6:关闭
hv_kvp_daemon   0:关闭  1:关闭  2:关闭  3:启用  4:关闭  5:启用  6:关闭
ip6tables       0:关闭  1:关闭  2:启用  3:启用  4:启用  5:启用  6:关闭
iptables        0:关闭  1:关闭  2:启用  3:启用  4:启用  5:启用  6:关闭
libvirt-guests  0:关闭  1:关闭  2:启用  3:启用  4:启用  5:启用  6:关闭
matahari-broker 0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
matahari-host   0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
matahari-network       0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
matahari-service       0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
matahari-sysconfig    0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
messagebus      0:关闭  1:关闭  2:启用  3:启用  4:启用  5:启用  6:关闭
mysqld         0:关闭   1:关闭  2:启用  3:启用  4:启用  5:启用  6:关闭
```

```
netconsole      0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
netfs           0:关闭  1:关闭  2:关闭  3:启用  4:启用  5:启用  6:关闭
network         0:关闭  1:关闭  2:启用  3:启用  4:启用  5:启用  6:关闭
nginxd          0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
ntpd            0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
ntpdate         0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
postfix         0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
pureftpd        0:关闭  1:关闭  2:启用  3:启用  4:启用  5:启用  6:关闭
qpidd           0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
rdisc           0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
restorecond     0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
rrdcached       0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
rsyslog         0:关闭  1:关闭  2:启用  3:启用  4:启用  5:启用  6:关闭
saslauthd       0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
snmpd           0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
snmptrapd       0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
sshd            0:关闭  1:关闭  2:启用  3:启用  4:启用  5:启用  6:关闭
sysstat         0:关闭  1:启用  2:启用  3:启用  4:启用  5:启用  6:关闭
udev-post       0:关闭  1:启用  2:启用  3:启用  4:启用  5:启用  6:关闭
wdapache        0:关闭  1:关闭  2:关闭  3:启用  4:关闭  5:启用  6:关闭
```

第 2 个用法，单独列出 ntpd 服务的设置情况：

```
[root@roclinux ~]# chkconfig --list ntpd
ntpd            0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
```

第 3 个用法，设定 ntpd 在等级 3 和 5 时为开启状态：

```
#开始设置，on表示启动，off表示关闭
[root@roclinux ~]# chkconfig --level 35 ntpd on

#检查一下，生效啦
[root@roclinux ~]# chkconfig --list ntpd
ntpd            0:关闭  1:关闭  2:关闭  3:启用  4:关闭  5:启用  6:关闭

#演示完成，再全部关掉，恢复到以前状态
[root@roclinux ~]# chkconfig ntpd off
[root@roclinux ~]# chkconfig --list ntpd
ntpd            0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
```

能讲讲 chkconfig 原理么

通过之前学习 service 命令，我们已经知道/etc/init.d/中包含了 Linux 系统的各种服务，其实运行等级（Runlevel）中的服务集合所对应的就是/etc/init.d/中的这些服务。

Linux 系统在/etc/rc.d/目录下设置了 7 个文件夹，即 rc0.d、rc1.d、rc2.d、rc3.d、rc4.d、

rc5.d 和 rc6.d，正好对应 7 个运行等级，我们来看一下：

```
#进入到/etc/rc.d文件夹
[root@roclinux ~]# cd /etc/rc.d/

#看看里面都有啥，看到那7个文件夹了吧
[root@roclinux rc.d]# ls -F
init.d/  rc*  rc0.d/  rc1.d/  rc2.d/  rc3.d/  rc4.d/  rc5.d/  rc6.d/
rc.local*  rc.sysinit*

#我们随便进到一个等级中
[root@roclinux rc.d]# cd rc3.d

#查看一下里面都有什么东西
[root@roclinux rc3.d]# ls -hl
总用量 0
lrwxrwxrwx.  1  root  root  23  2 月        7  2012  K01matahari-host
-> ../init.d/matahari-host
lrwxrwxrwx.  1  root  root  26  2 月       7  2012  K01matahari-network
-> ../init.d/matahari-network
lrwxrwxrwx.  1  root  root  26  2 月       7  2012  K01matahari-service
-> ../init.d/matahari-service
lrwxrwxrwx.  1  root  root  28  2 月      7  2012  K01matahari-sysconfig
-> ../init.d/matahari-sysconfig
lrwxrwxrwx  1 root root 19 11月  8 2014 K10rrdcached -> ../init.d/rrdcached
lrwxrwxrwx. 1 root root 19 2月   7 2012 K10saslauthd -> ../init.d/saslauthd
lrwxrwxrwx    1  root  root  22  11 月      8  2014  K15htcacheclean
-> ../init.d/htcacheclean
lrwxrwxrwx.  1  root  root  25  2 月       7  2012  K15matahari-broker
-> ../init.d/matahari-broker
lrwxrwxrwx  1 root root 16 6月  19 2012 K15nginxd -> ../init.d/nginxd
lrwxrwxrwx  1 root root 15 6月  14 2012 K15qpidd -> ../init.d/qpidd
lrwxrwxrwx  1 root root 17 6月  14 2012 K30postfix -> ../init.d/postfix
lrwxrwxrwx.  1  root  root  20  2 月        7  2012  K50netconsole
-> ../init.d/netconsole
lrwxrwxrwx  1 root root 15 1月  22 2014 K50snmpd -> ../init.d/snmpd
lrwxrwxrwx  1 root root 19 1月  22 2014 K50snmptrapd -> ../init.d/snmptrapd
lrwxrwxrwx  1 root root 14 2月  17 20:07 K74ntpd -> ../init.d/ntpd
lrwxrwxrwx. 1 root root 17 2月   6 2012 K75ntpdate -> ../init.d/ntpdate
lrwxrwxrwx.  1  root  root  21  2 月        7  2012  K87restorecond
-> ../init.d/restorecond
lrwxrwxrwx. 1 root root 15 2月   7 2012 K89rdisc -> ../init.d/rdisc
lrwxrwxrwx  1 root root 17 1月   6 2015 S01sysstat -> ../init.d/sysstat
lrwxrwxrwx. 1 root root 19 2月   7 2012 S08ip6tables -> ../init.d/ip6tables
lrwxrwxrwx. 1 root root 18 2月   7 2012 S08iptables -> ../init.d/iptables
lrwxrwxrwx. 1 root root 17 2月   7 2012 S10network -> ../init.d/network
lrwxrwxrwx. 1 root root 16 2月   7 2012 S11auditd -> ../init.d/auditd
lrwxrwxrwx. 1 root root 17 2月   7 2012 S12rsyslog -> ../init.d/rsyslog
```

```
lrwxrwxrwx.  1  root  root  20  2 月        7  2012  S22messagebus
-> ../init.d/messagebus
lrwxrwxrwx.  1  root  root  22  2 月       7  2012  S24avahi-daemon
-> ../init.d/avahi-daemon
lrwxrwxrwx. 1 root root 15 2月   7 2012 S25netfs -> ../init.d/netfs
lrwxrwxrwx. 1 root root 19 2月   7 2012 S26udev-post -> ../init.d/udev-post
lrwxrwxrwx    1  root  root  23  2 月     17  20:07  S50hv_kvp_daemon
-> ../init.d/hv_kvp_daemon
lrwxrwxrwx. 1 root root 14 2月   7 2012 S55sshd -> ../init.d/sshd
lrwxrwxrwx  1 root root 16 6月  19 2012 S64mysqld -> ../init.d/mysqld
lrwxrwxrwx  1 root root 15 11月  8 2014 S85httpd -> ../init.d/httpd
lrwxrwxrwx  1 root root 18 6月  19 2012 S85pureftpd -> ../init.d/pureftpd
lrwxrwxrwx  1 root root 18 6月  19 2012 S85wdapache -> ../init.d/wdapache
lrwxrwxrwx. 1 root root 15 2月   7 2012 S90crond -> ../init.d/crond
lrwxrwxrwx.  1  root  root  24  2 月       7  2012  S99libvirt-guests
-> ../init.d/libvirt-guests
lrwxrwxrwx. 1 root root 11 2月   7 2012 S99local -> ../rc.local
```

相当壮观啊！全部都是软链文件！

仔细一看，每一个都链接到了我们熟悉的/etc/init.d/文件夹中的某个服务文件上。

再仔细一看，所有软链文件的命令都有一定规律，都是以 K 开头或 S 开头，然后紧接着一个两位数的数字，最后是服务名。

如果大家和我观察的一样仔细，那么我想是时候揭晓答案了，chkconfig 的原理通过下面这几行文字就足以概括：

- /etc/init.d/中包含所有可用的服务；
- /etc/rc.d/中设置有 7 个文件夹，以 rcN.d 形式命名，分别对应 7 个运行等级；
- 每一个 rcN.d 文件夹中的文件全部是软链形式，分别链接到/etc/init.d/中的服务上，也就表明了当前的运行等级对应着哪些服务；
- rcN.d 中的软链文件命名规则；
  - ✧ K+整数+服务名；
  - ✧ S+整数+服务名。
- 以 K 开头的软链文件，表示要关闭对应的服务。以 S 开头的软链文件，表示要启动对应的服务。
- K、S 后面的数字，并非一定是两位数，完全可以是三位数，这个数字决定了各个服务的启动或关闭的前后顺序。但是其本质上并不是按照数字大小来排序，而是按照 ASCII 字符排序，比如

- ✧ S11 一定是在 S15 之前被启动的；
- ✧ S110 却是排在 S11 之后，但是在 S15 之前被启动；
- ✧ 若出现多个 S99，那么就会按照整个符号链接文件名来进行排序了。

6 dmidecode 看穿机器的底细

安装 dmidecode 工具

假如你的服务器上没有这个命令，那么用一条命令就可以安装好：

```
#RHEL、CentOS、Fedora
[root@roclinux ~]# yum install dmidecode

#Debian、Ubuntu
[root@roclinux ~]# apt-get install dmidecode

#Arch
[root@roclinux ~]# pacman -S dmidecode

#Gentoo
[root@roclinux ~]# emerge -av dmidecode
```

dmidecode 输出超多内容

首先，我们来看一下 dmidecode 的输出：

```
[root@roclinux ~]# dmidecode | wc -l
3064

[root@roclinux ~]# dmidecode | less
# dmidecode 2.12
SMBIOS 2.3 present.
216 structures occupying 17166 bytes.
Table at 0x000F8EC0.

Handle 0x0000, DMI type 0, 20 bytes
BIOS Information
       Vendor: American Megatrends Inc.
       Version: 090006
       Release Date: 05/23/2012
       Address: 0xF0000
       Runtime Size: 64 kB
       ROM Size: 256 kB
...
Handle 0x0005, DMI type 4, 35 bytes
Processor Information
```

```
        Socket Designation: None
        Type: Central Processor
        Family: Xeon
        Manufacturer: Intel
        ID: D7 06 02 00 FF FB 8B 1F
...
Handle 0x00D3, DMI type 17, 27 bytes
Memory Device
        Array Handle: 0x0051
        Error Information Handle: 0x0050
        Total Width: Unknown
        Data Width: Unknown
        Size: No Module Installed
...(此处省略上千行)
```

你会发现，dmidecode 的输出包含了 BIOS、CPU、内存等硬件信息，而在我的服务器上，dmidecode 足足输出了 3064 行。

我们可以猜到，dmidecode 一定是把这台服务器的各类硬件信息，一股脑的全都输了出来。真的有些不太友好啊。

我们想单独看某一类硬件的信息

dmidecode 支持哪些硬件呢，我们可以输入-t 选项，不加任何参数，就可以看到了。

```
[root@roclinux ~]# dmidecode -t
dmidecode: option requires an argument -- 't'
Type number or keyword expected
Valid type keywords are:
  bios
  system
  baseboard
  chassis
  processor
  memory
  cache
  connector
  slot
```

在-t 选项后面，加入我们想查看的某类硬件的名称，就可以查看到对应的信息啦！

比如我们想看 CPU 的硬件信息：

```
[root@roclinux ~]# dmidecode -t processor
# dmidecode 2.12
SMBIOS 2.3 present.

Handle 0x0005, DMI type 4, 35 bytes
Processor Information
        Socket Designation: None
        Type: Central Processor
```

```
        Family: Xeon
        Manufacturer: Intel
        ID: D7 06 02 00 FF FB 8B 1F
        Signature: Type 0, Family 6, Model 45, Stepping 7
...
...
```

再来看看内存的硬件信息：

```
[root@roclinux ~]# dmidecode -t memory
# dmidecode 2.12
SMBIOS 2.3 present.

Handle 0x0051, DMI type 16, 15 bytes
Physical Memory Array
        Location: Unknown
        Use: System Memory
        Error Correction Type: None
        Maximum Capacity: Unknown
        Error Information Handle: 0x0050
        Number Of Devices: 64
...
```

说到这里，其实大家已经知道了，dmidecode 的作用就是把各类硬件的信息输出出来而已。

但是，虽然大家知道了 dmidecode 的作用和用法，但是相信大家对 dmidecode 的输出内容还是毫无感觉，因为内容真的很晦涩，内容真的很冗长，对不对？

如果大家有兴趣了解更深一层的知识，那么，我们还是要从源头说起。

先说说什么是 DMTF

DMTF，全称是 Distributed Management Task Force，中文名称是分布式管理任务组，这是一个国际标准组织，里面的专家们都专注在“分布式 IT 系统的有效管理”这个课题上。

为什么要说 DMTF 呢，因为这个组织的前身是 Desktop Management Task Force，即桌面管理任务组，而接下来要介绍的 DMI 便是这个组织的杰作。

再说说什么是 DMI

我们为什么要介绍 DMI 呢，大家从 dmidecode 的名字应该能够看出一些线索。从字面理解的话，dmidecode 似乎就是针对 dmi 的一种解码工具。

哈哈，先不剧透，我们先来了解下 dmi 是什么东西。

DMI，是 Desktop Management Interface 的缩写，中文解释为桌面管理接口。

DMI 是一套统一的标准框架，用来管理和跟踪服务器、PC、笔记本等计算机的各个硬件。它的初期版本便是由 DMTF 来推进的。DMI 出现之后，让我们可以很方便地获取 PC 上各个硬件的详细信息。

早在 1999 年，微软便要求所有 OEM 厂商和 BIOS 提供商必须支持 DMI 标准，才能够获得微软的官方认证。这对 DMI 的推广起到了很大的作用。

DMI 和 SMBIOS 的渊源

SMBIOS，全称是 System Management BIOS，是一套数据结构标准，用来规范各个厂商的 BIOS 中信息的存储格式。大约是在 1999 年，SMBIOS 被纳入到 DMTF 的负责领域之中。但在这次整合之前，SMBIOS 的原名其实叫作 DMIBIOS。

之所以叫 DMIBIOS，就是因为在设计之初，它就是被确定为与 DMI 交互的数据结构标准。

更通俗地讲，SMBIOS 规范了 BIOS 中的数据的存储格式，而 DMI 则是读取 BIOS 中这类数据的一套框架或工具。

那么，说到这里，我们就要来介绍今天的男一号 dmidecode 啦，dmidecode 是 Linux 系统中实现 DMI 的具体工具。

dmidecode 才是正题

在介绍了那么多背景知识和名词之后，在下面的内容里，我们就可以放心地提到它们了，因为大家现在再阅读到这些名词，就不会感到那么晦涩和茫然了。

好，我们开始。

dmidecode 工具，可以获取 Linux 系统中的各类硬件信息，它遵循 SMBIOS/DMI 标准，可以输出的硬件信息如下：

- BIOS
- 系统
- 主板
- 机箱
- 处理器
- 内存
- 缓存
- 连接器
- 插槽

是不是很全面，以后再也不用担心我们查不到硬件信息了。

dmidecode 所作的工作，其实并不算复杂，它负责将 SMBIOS/DMI 格式的信息解码后，以用户可读的格式展示出来。

不过，要留意的一点是，和汽车发动机的编号一样，SMBIOS/DMI 数据也是完全可以人为修改或捏造的，所以不能完全地相信它所输出的一些序列号和版本号等信息。但通常情况下，没有人会去刻意作假。

dmidecode 的两个选项

如果直接执行 dmidecode 命令的话，会将所有的硬件信息都输出出来，大概会有三千多行的样子。这种“海量的”输出形式，只适合于机器处理，不太适合用户阅读。

所以我们要学会使用 dmidecode 的一些选项，来更精准地找出自己的所需。

dmidecode 最常使用的选项有两个，都是用来寻找精准信息的，即-t 选项和-s 选项。

-t 选项，即--type 选项，用来指定要展示哪类硬件信息，这样就可以有效地缩小范围了，支持数字形式和字符串形式。

目前 dmidecode 支持的字符串形式如表 26 所示。

表 26 dmidecode 支持的字符串形式

类型名	中文名
BIOS	BIOS
system	系统
baseboard	主板
chassis	机箱

续表

类型名	中文名
processor	处理器
memory	内存
cache	缓存
connector	连接器
slot	插槽

目前支持的数字形式如表 27 所示。

表 27　支持的数字形式

类型编号	对应信息	类型编号	对应信息
0	BIOS	20	Memory Device Mapped Address
1	System	21	Built-in Pointing Device
2	Base Board	22	Portable Battery
3	Chassis	23	System Reset
4	Processor	24	Hardware Security
5	Memory Controller	25	System Power Controls
6	Memory Module	26	Voltage Probe
7	Cache	27	Cooling Device
8	Port Connector	28	Temperature Probe
9	System Slots	29	Electrical Current Probe
10	On Board Devices	30	Out-of-band Remote Access
11	OEM Strings	31	Boot Integrity Services
12	System Configuration Options	32	System Boot
13	BIOS Language	33	64-bit Memory Error
14	Group Associations	34	Management Device
15	System Event Log	35	Management Device Component
16	Physical Memory Array	36	Management Device Threshold Data
17	Memory Device	37	Memory Channel
18	32-bit Memory Error	38	IPMI Device
19	Memory Array Mapped Address	39	Power Supply

我们拿 Chassis 来举例：

```
[root@roclinux ~]# dmidecode -t chassis
# dmidecode 2.12
```

```
SMBIOS 2.3 present.

Handle 0x0003, DMI type 3, 17 bytes
Chassis Information
    Manufacturer: Microsoft Corporation
    Type: Desktop
    Lock: Not Present
    Version: 7.0
    Serial Number: 0057-3958-2964-7105-9370-5910-56
    Asset Tag: 8225-4594-6962-7440-7277-2623-99
    Boot-up State: Safe
    Power Supply State: Safe
    Thermal State: Other
    Security Status: Other
    OEM Information: 0x00000000
```

而-s 选项，即--string 选项，显示指定的 DMI 字段的信息。这个选项要比-t 选项更加精准，要求用户指明具体的字段来显示，目前支持的字段如表 28 所示。

表 28 -s 选项支持的字段

字段名	中文名	字段名	中文名
bios-vendor	BIOS 提供商	baseboard-serial-number	主板序列号
bios-version	BIOS 版本号	baseboard-asset-tag	主板资产标记
bios-release-date	BIOS 发布日期	chassis-manufacturer	机箱制造商
system-manufacturer	系统制造商	chassis-type	机箱类型
system-product-name	系统产品名	chassis-version	机箱版本
system-version	系统版本	chassis-serial-number	机箱序列号
system-serial-number	系统序列号	chassis-asset-tag	机箱资产标记
system-uuid	系统 UUID	processor-family	处理器系列品牌
baseboard-manufacturer	主板制造商	processor-manufacturer	处理器制造商
baseboard-product-name	主板产品名	processor-version	处理器版本
baseboard-version	主板版本	processor-frequency	处理器频率

我们该如何使用 dmidecode 呢

dmidecode 中包括了各类硬件的信息，每一个厂商、每一类硬件都有它特有的知识和参数，如果要深入解读的话，或许又可以再写好几本书了。

所以呢，我们只在这里举一个小例子，也是大家很常见的，就是查看当前设备能支持的最大内存，以及当前我们的内存插槽情况：

```
[root@roclinux ~]# dmidecode -t 16
# dmidecode 2.12
SMBIOS 2.4 present.
```

```
Handle 0x1000, DMI type 16, 15 bytes
Physical Memory Array
    Location: Other
    Use: System Memory
    Error Correction Type: Multi-bit ECC
    Maximum Capacity: 4 GB
    Error Information Handle: Not Provided
    Number Of Devices: 1
```

我们使用的类型值是 16，表示的是“Physical Memory Array”，即物理内存阵列信息。从中可以看出，最大内存额（Maximum Capacity）是 4GB，也就是说本设备最大可以支持的内存就是 4GB。

如果你有兴趣，还可以查看一下当前设备的内存条插槽情况，然后就可以制定出你的内存条扩容方案啦。这些内容，我们就不在这里展开了：

```
[root@roclinux ~]# dmidecode -t 17
# dmidecode 2.12
SMBIOS 2.4 present.

Handle 0x1100, DMI type 17, 21 bytes
Memory Device
    Array Handle: 0x1000
    Error Information Handle: 0x0000
    Total Width: 64 bits
    Data Width: 64 bits
    Size: 4096 MB
    Form Factor: DIMM
    Set: None
    Locator: DIMM 0
    Bank Locator: Not Specified
    Type: RAM
    Type Detail: None
```

7 lsmod 列出内核模块

Linux 内核的知识

Linux 内核是宏内核（和微内核相对），整个内核是一个单独的、非常庞大的程序。在这种结构中，硬件驱动的维护是相当麻烦的，添加和删除硬件驱动时，都需要重新编译内核才能生效。

如今市场上的硬件真是日新月异，几乎每天都有新的产品出现。为了应对这种变化，Linux 引入了一种称为 module（模块）的技术，它可以实现把某些功能代码封装成模块动态地装载到内核中，当内核需要用到这个功能时再读取使用。这种技术被广泛应用到了硬件驱动开发中，大大改善了 Linux 内核对新硬件的支持能力。

内核与内核模块放在哪

内核与内核模块的位置如表 29 所示。

表 29　内核与内核模块的位置

名　　称	位　　置
内核	/boot/vmlinuz 或 /boot/vmlinuz-version
内核解压缩所需的 RAMDisk	/boot/initrd 或 /boot/initrd-version
内核模块	/lib/modules/version/kernel 或 /lib/modules/$(uname -r)/kernel
内核源码	/usr/src/Linux 或 /usr/src/kernels

如果想查看内核版本，可以直接查看/proc/version 文件，或者使用 uname 命令。而如果想查看或设置内核参数，则需要到/proc/sys/kernel 目录中去。

内核模块分类管理

我们知道市面上硬件众多，驱动模块也多如牛毛，Linux 是如何有条理地管理它们

的呢？

这是一个好问题，我们可以到系统中的/lib/modules/$(uname -r)/kernel 目录下去寻找答案：

```
[roc@roclinux ~]$ ls -1F /lib/modules/2.6.32-220.4.1.el6.x86_64/kernel/
arch/
crypto/
drivers/
fs/
lib/
mm/
net/
sound/
```

可见，Linux 是将模块分成若干种分类来分别管理的，这些分类如表 30 所示。

表 30　模块的种类

分类名	描　　述
arch	与硬件平台有关的项目，例如 CPU 的等级等
crypto	核心所支持的加密的技术，例如 md5 或者 des 等
drivers	一些硬件的驱动程序，例如显卡、网卡、PCI 相关硬件等
fs	核心所支持的文件系统，例如 vfat、reiserfs、nfs 等
lib	一些函数库
mm	内存管理相关
net	与网络有关的各项协议数据
sound	与音效有关的各项模块

这样就一目了然、清清楚楚地知道模块的位置了吧。

内核模块的依赖关系

问题又来了，模块这么多，它们交错在一起协同工作，依赖关系又成了难题（呵呵，Linux 中的依赖关系可是一个难点），如果要手动一个一个地去查看这些模块，然后定义出它们的依赖关系，我们可能会疯掉吧！不用担心，Linux 早已提供了一个模块依赖关系的解决方案，查看 /lib/modules/$(uname -r)/modules.dep 这个文件，它记录了内核所支持的所有模块的依赖关系。

我们来看看这个文件的内容，每一行表示一个依赖关系：

```
[roc@roclinux ~]$ cat /lib/modules/2.6.32-220.4.1.el6.x86_64/modules.dep
kernel/arch/x86/kernel/cpu/mcheck/mce-inject.ko:
kernel/arch/x86/kernel/cpu/cpufreq/powernow-k8.ko:
kernel/drivers/cpufreq/freq_table.ko
```

```
kernel/arch/x86/kernel/cpu/cpufreq/mperf.ko
kernel/arch/x86/kernel/cpu/cpufreq/mperf.ko:
kernel/arch/x86/kernel/cpu/cpufreq/acpi-cpufreq.ko:
kernel/drivers/cpufreq/freq_table.ko
kernel/arch/x86/kernel/cpu/cpufreq/mperf.ko
kernel/arch/x86/kernel/cpu/cpufreq/pcc-cpufreq.ko:
kernel/arch/x86/kernel/cpu/cpufreq/speedstep-lib.ko:
kernel/arch/x86/kernel/cpu/cpufreq/p4-clockmod.ko:
kernel/drivers/cpufreq/freq_table.ko
kernel/arch/x86/kernel/cpu/cpufreq/speedstep-lib.ko
kernel/arch/x86/kernel/test_nx.ko:
kernel/arch/x86/kernel/microcode.ko:
kernel/arch/x86/crypto/fpu.ko:
kernel/arch/x86/crypto/aes-x86_64.ko: kernel/crypto/aes_generic.ko
（此处省略数百行）
```

那么，这个文件又是怎么来的呢？很简单！利用 depmod 这个命令就可以创建出这个文件。

比如我们写好了一个网卡驱动程序，文件名为 netdriver.ko，我们该如何把它加入到内核模块的依赖关系中呢？简单的两条命令就可以搞定：

```
[root@roclinux ~]# cp driver.ko /lib/modules/$(uname -r)/kernel/drivers/net
[root@roclinux ~]# depmod
```

当 depmod 命令运行完成之后，网卡驱动程序文件会被自动放到模块目录 /lib/modules/$(uname -r)/kernel 中，与此同时，depmod 还会依据相关目录的定义分析全部的内核模块，并将内核模块的依赖关系写入 modules.dep 文件。这是多么体贴周到的服务啊！

显示所有的内核模块

那我能不能知道系统中现在使用的都有哪些模块呢？不会让我去查看 modules.dep 文件吧，那太麻烦了。不用担心，Linux 还为大家准备了 lsmod 命令：

```
[root@roclinux ~]# lsmod
Module                  Size  Used by
autofs4                24517  2
hidp                   23105  2
(中间省略数十行)
8139too                28737  0
8139cp                 26305  0
mii                     9409  2 8139too,8139cp ≤==mii模块还被8139cp、8139too
所引用
(中间省略数十行)
uhci_hcd               25421  0  ≤==底下三个是优盘相关的模块!
ohci_hcd               23261  0
```

```
ehci_hcd               33357  0
```

lsmod 命令的输出中包括了四列内容，它们分别是：

- 第一列：模块名称。
- 第二列：模块大小。
- 第三列：被引用的数量。如果后面有 autoclean，则表示该模块可以在空闲时自动卸载。如果后面有 unused，则表示该模块当前没在使用。同时有两个标识的话，可以直接被 rmmod -a 命令自动卸载。
- 第四列：表示此模块被哪些模块所引用。

聚焦到某一模块

当我们要把目光聚焦到某一个模块时，可以使用 modinfo 命令来查看某个模块的具体信息：

```
[root@roclinux ~]# modinfo mii
filename:       /lib/modules/2.6.18-92.el5/kernel/drivers/net/mii.ko
license:        GPL
description:    MII hardware support library
author:         Jeff Garzik ≤jgarzik@pobox.com>
srcversion:     16DCEDEE4B5629C222C352D
depends:
vermagic:       2.6.18-92.el5 SMP mod_unload 686 REGPARM 4KSTACKS gcc-4.1
```

通过 modinfo 命令，还可以看到这个模块的作者、版权、存放位置以及该模块的简介。

从 description: MII hardware support library 这个信息来看，原来这个模块是硬件支持的函式库。

好了，我们在本文中为大家介绍了围绕 lsmod 命令的各种背景知识和相关命令，有兴趣学习和研究 Linux 内核的同学，可以以本文为入门，开启你的内核学习之旅吧！

8 最古老的容器技术 chroot

chroot 是什么

话说 2016 年 IT 技术圈最火的技术是什么？当然是容器了，这一年 Docker 技术真可谓如日中天，无人能敌，但这哥们太新太牛太复杂了，对于一些 Linux 玩家来说，还是有点难度。不过，容器这项技术并不稀奇，在 Linux 中很早就出现了，其中，最古老的要数 chroot 了。今天我们来点怀旧风，一起来看看 chroot 的那些事儿。

chroot，即 change root directory（更改 root 目录）。在 Linux 系统中，系统默认的目录结构是 /，即是以根（root）开始的。而在使用 chroot 之后，事情就发生了变化，系统的目录结构将以新指定的位置作为系统的根目录。

初识 chroot

chroot 如何使用呢？我们还是实际操作一下吧。

```
[root@roclinux chroot]# mkdir newroot
[root@roclinux chroot]# chroot newroot/
chroot: failed to run command `/bin/bash': No such file or directory
```

为什么上面的命令执行会出错？而且命令提示“failed to run command'/bin/bash': No such file or directory”？

这是因为 chroot 到一个新的目录后，新的目录就会变成文件系统的根目录，接着会执行新目录下的/bin/bash 程序。而我们的新建目录下并没有这个程序，所以执行报错了。

```
[root@roclinux chroot]# mkdir –p newroot/bin newroot/lib newroot/lib64
[root@roclinux chroot]# cp -r /bin/* ./newroot/bin/
[root@roclinux chroot]# cp -r /lib/* ./newroot/lib/
[root@roclinux chroot]# cp -r /lib64/* ./newroot/lib64/
[root@roclinux chroot]# chroot newroot/
bash-4.1# ls
bin  lib  lib64
bash-4.1# pwd
```

```
/
bash-4.1#
```

我们把系统中根目录下的 bin/lib/和 lib64/目录以及其下的所有文件都复制到新建目录下，然后 chroot，这次命令执行成功，并且新建目录变成了新的文件系统的根目录。

chroot 能干什么

chroot 可以在原有系统中创建一个独立的虚拟化的系统。可以用来做以下工作。

- 测试和开发

  可以在虚拟化系统中测试软件，减少在生产环境中测试带来的风险。

- 依赖控制

  可以在虚拟化系统中安装需要的依赖关系，并进行软件的开发、构建和测试。这可以防止开发人员安装过多的不同的程序库。

- 兼容性

  软件有时必须运行在 chroot 虚拟环境中， 因为软件使用的库或数据文件可能和主机系统发生了名称或链接冲突。

- 修复

  系统无法开机时，可以使用 chroot 引导另一个根文件系统（如 Live CD），然后修复原来损坏的环境。

- 特权分离

  chroot 的虚拟化环境就是一个程序沙盒，可以用来隔离特权用户的操作对原有系统的破坏。

详解 chroot 创建空间

需求：在虚拟主机上创建一个与原系统完全隔离的虚拟目录空间。

下面就用最原始的武器 chroot 命令来实现。

第一步：创建虚拟目录。

```
[roc@roclinux ~]$ mkdir newroot
```

第二步：创建命令执行的环境。

```
[roc@roclinux ~]$ mkdir -p newroot/{bin,lib,lib64}
[roc@roclinux ~]$ tree
.
└── newroot
    ├── bin
    ├── lib
    └── lib64

4 directories, 0 files
```

第三步：生成可执行的命令。

```
[roc@roclinux ~]$ cd newroot/
[roc@roclinux newroot]$ cp -v /bin/{bash,ls} ./bin/
`/bin/bash' -> `./bin/bash'
`/bin/ls' -> `./bin/ls'
[roc@roclinux newroot]$ cp -v /usr/bin/git ./bin
`/usr/bin/git' -> `./bin/git'
[roc@roclinux newroot]$ tree
.
├── bin
│   ├── bash
│   ├── git
│   └── ls
├── lib
└── lib64

3 directories, 3 files
[roc@roclinux ~]$ cd ..
```

第四步：切换到 root 用户，执行 chroot。

```
[root@roclinux ~]# pwd
/root
[root@roclinux ~]# tree
.
└── newroot
    ├── bin
    │   ├── bash
    │   ├── git
    │   └── ls
    ├── lib
    └── lib64

4 directories, 3 files
[root@roclinux ~]# chroot newroot/
chroot: failed to run command `/bin/bash': No such file or directory
```

在新建目录下不是已经有./bin/bash 程序了吗？为什么还会出现上面的错误呢？这是因为 bash 在新的文件系统环境下，无法找到程序使用的动态库，让我们先查看

一下 bash 程序都需要哪些动态库。

```
[root@roclinux ~]# ldd ./newroot/bin/bash
    Linux-vdso.so.1 =>  (0x00007fffd9d7d000)
    libtinfo.so.5 => /lib64/libtinfo.so.5 (0x00007fb85a6f5000)
    libdl.so.2 => /lib64/libdl.so.2 (0x00007fb85a4f1000)
    libc.so.6 => /lib64/libc.so.6 (0x00007fb85a15c000)
    /lib64/ld-Linux-x86-64.so.2 (0x00007fb85a91e000)
```

把上面的动态库复制到新文件系统目录下：

```
[root@roclinux ~]# cp /lib64/libtinfo.so.5 ./newroot/lib64/
[root@roclinux ~]# cp /lib64/libdl.so.2 ./newroot/lib64/
[root@roclinux ~]# cp /lib64/libc.so.6 ./newroot/lib64/
[root@roclinux ~]# cp /lib64/ld-Linux-x86-64.so.2 ./newroot/lib64/
[root@roclinux ~]# chroot newroot/
bash-4.1#
```

执行 chroot 成功。

第五步：执行 Shell 命令。

chroot 成功，先来看看目录下都有什么文件吧。

```
bash-4.1# ls
ls: error while loading shared libraries: libseLinux.so.1: cannot open
shared object file: No such file or directory
bash-4.1#
```

怎么还是出错呢？这是因为 ls 命令使用了动态链接库 libseLinux.so.1，在 chroot 后，由于动态库不在新的执行环境中，因此会报出上面的错误。解决的方法同上面的 bash 方法：

```
[root@roclinux ~]# ldd /bin/ls
    Linux-vdso.so.1 =>  (0x00007fff49165000)
    libseLinux.so.1 => /lib64/libseLinux.so.1 (0x00007fd19c1a2000)
    librt.so.1 => /lib64/librt.so.1 (0x00007fd19bf9a000)
    libcap.so.2 => /lib64/libcap.so.2 (0x00007fd19bd95000)
    libacl.so.1 => /lib64/libacl.so.1 (0x00007fd19bb8d000)
    libc.so.6 => /lib64/libc.so.6 (0x00007fd19b7f9000)
    libdl.so.2 => /lib64/libdl.so.2 (0x00007fd19b5f4000)
    /lib64/ld-Linux-x86-64.so.2 (0x00007fd19c3cd000)
    libpthread.so.0 => /lib64/libpthread.so.0 (0x00007fd19b3d7000)
    libattr.so.1 => /lib64/libattr.so.1 (0x00007fd19b1d2000)
```

然后使用下列命令把所有的动态库复制到./gitRoom/lib64/下即可。

```
[root@roclinux ~]# list="$(ldd /bin/ls | egrep -o '/lib64.*\.[0-9]')"
[root@roclinux ~]# for i in $list; do cp "$i" "./newroot${i}"; done
[root@roclinux ~]# chroot newroot/
bash-4.1# ls
bin  lib  lib64
```

第六步：生成更多的可执行命令。

前面只是移植了 ls 的命令，如果想移植更多的命令，就可以使用上述方法。

但太烦琐了，下面来一个一劳永逸的方法。

```
[root@roclinux ~]# cp /bin/* ./newroot/bin/
[root@roclinux ~]# cp /lib64/* ./newroot/lib64/
[root@roclinux ~]# chroot newroot/
bash-4.1# ls
bin  lib  lib64
bash-4.1# cd bin/
```

好了，原系统中/bin 的所有命令都可以在虚拟目录下执行了。

至此，我们创建的虚拟目录基本成功，如果你有新命令的需求，就请采用上面的方法来解决吧。

9 玩转关机和重启

乱花渐欲迷人眼

很多人都喜欢研究 Linux 的启动环节，但很少有人会对 Linux 的关机环节感兴趣。其实关机环节，也有很多的知识点可以了解和学习。

仅仅是和关机、重启有关的命令，就有至少五个，相信你已经傻傻分不清楚啦：

- shutdown
- reboot
- halt
- poweroff
- init

可能很少有人能清晰地说出他们的关系和区别吧！

走马观花

吊胃口的感觉，总是很难受的，所以呢，我们先来快速了解下这几个命令的作用和用法，如表 31 所示。

表 31　作用和用法

命　令	作　用
shutdown	可用于关机、重启，支持定时和通知
reboot	重启系统
halt	停止系统
poweroff	关机
init	init 0 用于关机，init 6 用于重启

我们继续来看它们的实际用法：

```
#用shutdown立即关机
$shutdown -h now
```

```
#用shutdown来立即重启
$shutdown -r now

#我想用在晚上23:30设置定时关机
$shutdown -h 23:30

#我想在15分钟后关机
$shutdown -h +15

#我想用halt关机并关闭电源
$halt -p

#我想关机，但不在日志里留下任何关机的痕迹
$halt -d

#不真关机，只是把关机的事件记录到/var/log/wtmp中
$halt -w

#我要重启系统
$reboot

#我想关机并切断电源
$poweroff

#我想重启
$init 6

#我想关机
$init 0
```

大致了解了每个命令的作用和用法之后，我猜你的疑惑来了，既然都是实现关机或重启，那么 Linux 为什么要提供这么多不同的选择呢？到底哪一种关机方式才是最安全的呢？

带着问题，我们继续向下阅读，相信读到文章结束时，你一定会有更深入的理解和认识。

reboot、halt 和 poweroff 一家亲

说它们是一家，并不是因为它们的作用相同，而是因为它们本来就是同一个程序：

```
#找到reboot、halt、poweroff的文件位置
[root@roclinux roc]# which reboot
/sbin/reboot
[root@roclinux roc]# which halt
/sbin/halt
[root@roclinux roc]# which poweroff
```

```
/sbin/poweroff

#通过md5值计算，发现他们完全相同
[root@roclinux roc]# md5sum /sbin/reboot /sbin/halt /sbin/poweroff
f68eec63306a6073f3e489fb2ed0f172  /sbin/reboot
f68eec63306a6073f3e489fb2ed0f172  /sbin/halt
f68eec63306a6073f3e489fb2ed0f172  /sbin/poweroff

#再查看文件类型，原来三者是软链接关系，最终都指向了reboot程序
[root@roclinux roc]# ls -hl /sbin/reboot /sbin/halt /sbin/poweroff
lrwxrwxrwx. 1 root root   6 2月   7 2012 /sbin/halt -> reboot
lrwxrwxrwx. 1 root root   6 2月   7 2012 /sbin/poweroff -> reboot
-rwxr-xr-x. 1 root root 15K 7月  19 2011 /sbin/reboot
```

但是，我很负责任地告诉你，虽然三者本质上是一个文件，但是你在执行它们的时候，三者的表现是不同的。这又是为什么呢？

这里引入一个小插曲。其实这是 Linux 里的一种常见玩法，就是把代码逻辑大体相同的程序融合成一个程序文件，然后通过在代码中判断程序调用名来进行不同分支的处理，比如 gzip 和 gunzip、pidof 和 killall5 都是如此。

为了让同学们理解得足够充分，这里特地把最关键的那段源代码也为同学们展示了出来：

```
  mode = REBOOT;
  if (! strcmp (program_name, "halt")) {
    mode = HALT;
    nih_option_set_synopsis (_("Halt the system."));
  } else if (! strcmp (program_name, "poweroff")) {
    mode = POWEROFF;
    nih_option_set_synopsis (_("Power off the system."));
  } else {
    mode = REBOOT;
    nih_option_set_synopsis (_("Reboot the system."));
  }
```

ACPI 的故事

不是要讲上面几个命令的区别么，怎么突然就跳到 ACPI 了呢？这是什么啊？

其实这样安排是有我们的特别用意的，因为后面的讲解中会出现 ACPI 这个概念，为了让大家能够更好地阅读和理解，我们有必要提前做好关于这个概念的铺垫工作，否则的话，一堆新名词突然出现，想必会很影响大家的阅读体验的。

好了，闲言少叙，我们先来了解第一个新概念 ACPI。

ACPI，英文全称是 Advanced Configuration and Power Interface，中文全称是“高级配置和电源接口标准”，好晦涩的名称啊。

简单地说，电源管理是系统中的一个重要模块，它负责将电能更有效地分配给系统中的不同组件。一款优秀的电源管理模块，可以将电池寿命延长到原来的两到三倍。电源管理需要涉及的方面非常广，包括但不限于：

1. 处理器电源管理
2. 系统电源管理
3. 设备电源管理
4. 即插即用
5. 系统事件
6. 电池管理
7. 温度管理

而所谓电源接口，就是提供了针对上述各个方面电源管理的功能接口。

ACPI，便是制定了一整套有关电源管理接口的工业标准，涉及 OS 管理、电源管理和温度管理，服务对象也覆盖到了移动设备、桌面电脑和服务器平台。

ACPI 这套标准，是在 1997 年由惠普、英特尔、微软、菲尼克斯电器和东芝共同制定的，目前发展到了 5.0a 版本。

在 ACPI 之前，曾经广泛使用的是 APM（高级电源管理）。由于 APM 是由 BIOS 来控制和实现的，这就造成了 APM 在兼容性和对 USB 等新硬件的支持上的缺陷，也一定程度上影响了能耗控制的效果。而 ACPI 则是实现在操作系统这一层，这让 ACPI 具有了更好的可扩展性和灵活性，也可以实现诸如休眠这样的效果。

我的机器上用的是哪套电源管理标准

我想查看一下 CentOS 服务器上用的是什么电源管理标准：

```
[root@roclinux ~]# dmesg |grep -i acpi
 BIOS-e820: 00000000c17f0000 - 00000000c17ff000 (ACPI data)
 BIOS-e820: 00000000c17ff000 - 00000000c1800000 (ACPI NVS)
ACPI: RSDP 00000000000f56f0 00014 (v00 ACPIAM)
ACPI: RSDT 00000000c17f0000 00040 (v01 DELL   PE_SC3   05001223 MSFT
00000097)
ACPI: FACP 00000000c17f0200 00081 (v02 DELL   PE_SC3   05001223 MSFT
```

```
00000097)
ACPI: DSDT 00000000c17f1724 02E78 (v01 MSFTVM MSFTVM02 00000002 INTL
02002026)
ACPI: FACS 00000000c17ff000 00040
(此处省略数百行)
PCI: Using ACPI for IRQ routing
pnp: PnP ACPI init
ACPI: bus type pnp registered
pnp: PnP ACPI: found 13 devices
ACPI: ACPI bus type pnp unregistered
acpiphp: ACPI Hot Plug PCI Controller Driver version: 0.5
ACPI: Power Button [PWRF]
ACPI: acpi_idle registered with cpuidle
pata_acpi 0000:00:07.1: setting latency timer to 64
```

从系统加载阶段输出的信息可以看出，系统使用的是 ACPI，而非 APM。即使你在有些系统上看到了 APM 的字样，也往往是被 ACPI 替代的信息：

```
[root@roclinux ~]# dmesg |grep -i apm
apm: overridden by ACPI。
```

如果我们查看系统进程，也会发现 ACPI 的蛛丝马迹：

```
[root@roclinux ~]# ps auxww|grep acpi
root       38  0.0  0.0     0    0 ?       S    2015   0:00 [kacpid]
root         39   0.0   0.0       0      0 ?         S      2015    0:00
[kacpi_notify]
root         40   0.0   0.0       0      0 ?         S      2015    0:00
[kacpi_hotplug]
```

看，这里的 kacpid 就表示 Kernel ACPI Daemon，也证明了我们的系统采用的是 ACPI。

了解一下 wtmp 和 utmp

讲完 ACPI，我们还有必要了解一下 wtmp 和 utmp 这两个文件，因为这两个文件也参与到了关机和重启的环节之中。

wtmp 文件，位于/var/log/下，是一个二进制形式的日志文件，该日志文件记录了每个用户登录、注销及系统的启动、停机事件。

我们通常所调用的 last 命令，就是读取 wtmp 文件得来的：

```
[roc@roclinux ~]$ last
roc      pts/0       221.239.109.12   Sun Feb 28 10:59   still logged in
yangwenq pts/0        114.241.157.47   Sat Feb 27 16:47 - 18:08  (01:20)
roc      pts/4       114.241.157.47   Sat Feb 27 13:48 - 18:07  (04:19)
roc      pts/0       114.241.157.47   Sat Feb 27 10:19 - 14:04  (03:45)
yangwenq pts/4        221.223.138.250  Fri Feb 26 11:39 - 21:28  (09:48)
```

```
roc      pts/0        221.223.138.250  Fri Feb 26 11:07 - 20:43  (09:36)
roc      pts/0        114.225.94.204   Thu Feb 25 16:32 - 00:06  (07:34)
yangwenq pts/4         114.225.94.204   Thu Feb 25 11:59 - 15:03  (03:04)
(如下省略数百行)
```

utmp 文件，则位于/var/run/下，同样是一个二进制形式的文件，它记录的是当前正在本系统中的用户的信息：

```
[roc@roclinux ~]$ w
 11:14:44 up 108 days, 18:28,  1 user,  load average: 0.53, 0.54, 0.36
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
roc      pts/0    221.219.52.53    10:59    0.00s  0.33s  0.00s w
```

犹抱琵琶，进入主题

前面我们花了太多篇幅来介绍命令的基本功能和用法、电源管理和 ACPI、wtmp/utmp 的作用，终于，我们把背景知识都介绍完了，下面就可以正式介绍几个关机命令的区别啦。

对于 shutdown 命令，你应该了解的秘密是：

- shutdown -r 用来重启，它基本等价于 reboot。
- shutdown -h -P 用来实现停止系统和关闭电源，它基本等价于 poweroff，而 poweroff 完全等价于 halt -p。
- shutdown -h 则用来实现停止系统，但不关闭电源，它基本等价于 halt。
- 可以看出 shutdown 命令是可以由 reboot/halt 来替代的。
- 而 shutdown 与 reboot/halt 的最主要区别在于，shutdown 会给登录系统的用户发送关机或重启的消息通知，这或许是 shutdown 的唯一优势了。
- 如果只想给登录系统的用户发送关机或重启的消息，并不想真正重启或关机，那么请使用 shutdown -k。
- 如果给 shutdown 指定了一个非当前的时间点，则 shutdown 在发送完通知消息后会设置一个 timer_callback 回调，等待时间到来时再触发具体的关机或重启动作。

对于 init，你应该知道的是这些：

- init 的具体行为是依赖于本机 init 系统的，即主流的 SysvInit、UpStart 和 Systemd 之一。
- 通常情况下，关机的运行等级（runlevel）为 0；重启的运行等级为 6。

- 在运行等级为 0 或 6 的状态下，init 会进行这些主要动作，包括各类进程的停止、SWAP 的关闭、文件系统的卸载、quota 配额的关闭、UPS 系统的关闭，最后调用 halt 或 reboot 实现关机或重启。

由此可见，无论是 shutdown 还是 init，最终都会对应到 halt/reboot 命令上来，那我们就来一起了解下 reboot/halt 的内幕吧！

- halt 命令本身（不加-p 选项的话）表示停止系统，但并不包括切断电源，需要人工关闭电源开关，就好像很早以前的 Windows95 一样。
- halt -p 表示停止系统且切断电源。切断电源这一步会通过调用 ACPI 电源管理接口来实现。
- 如果 reboot/halt 使用了-w 选项，则只会将关机或重启的事件写入到 utmp/wtmp 文件中，并不会真正关机或重启。
- 通过上面的描述，我们可以看到所有的关机和重启都可以收敛到 reboot 和 halt 两个命令上。而 halt 和 reboot 其实最终调用的都是 reboot 系统调用，来实际完成关机或重启的动作。

如果你仔细阅读了上面的这些内容，相信你已经清楚了这些关机命令之间的区别了。如果必须用简短的话语再做一个总结的话，我想应该是这样：

- 请使用 shutdown -r 来重启，请使用 shutdown -h -P 来关机。
- shutdown 和 poweroff 可以由 reboot 和 halt 替代。
- reboot 和 halt 则最终调用 reboot 系统调用来实现关机和重启。

轻松一下，我们来个恶作剧吧！

刚才说了那么多晦涩难懂的知识，相信大家已经有些晕了，现在呢，我们需要换一下脑子了，搞一个恶作剧吧！

```
$shutdown -k now "Server would shutdown in 15minutes\!"
```

相信登录到机器上工作的用户，都会来敲你家门的哦，哈哈。

致谢

对于本书，我首先要感谢写作团队的杨文强、张昱两位作者，写书的过程漫长且孤独，正是因为有了他们的支持和信任，才让我们的这本书得以成型。

作者杨文强，是安全领域的专家，他踏实、严谨、一丝不苟，是一位少说多做的实干家。他的快速学习能力、总结能力、细节技术的探究能力，都是值得我学习的地方。

另一位作者张昱，长期从事数据库技术研究，他灵光聪明、开朗健谈，总能让我们写作团队在单调的写作过程中享受到欢乐。另外他总坚持查阅 Linux 英文技术资料，这也让我们总能获得更深度更正确的知识解读。

无论如何，对家人的感谢都是最重要的，感谢我的奶奶、我的父母、我的岳父岳母、我的妻子和孩子，以及我的亲人们，是你们让我有了精神的寄托，有了前进的动力，是你们默默承担起了繁重的家务，让我可以更加专注地写作。

学生时代，是梦想之花播种的年代，要感谢我的老师们，感谢高中班主任杜方老师、初中班主任杨春芝老师、初中语文教师刘淑珍老师、小学班主任尹和老师、小学数学老师李文兰老师，和已经仙逝的大学班主任张大利老师，是你们在我幼小的心灵中给予了阳光和雨露，让我朝着正确的方向进步和成长。

硕士生涯，是对一个人科研能力和工程能力的最佳培养期。感谢我的导师马严教授，他在学术上的辉煌成就、在做事上的严谨态度、在教育上的循循善诱，都是我敬重和学习的榜样。他在 IPv6 技术、移动 IP 技术、网络管理与网络安全技术上的科研成果，也是我无法企及的高度。祝愿我的导师马严教授身体健康，一切都好。还要感谢黄小红副教授，是您对我的研究课题给予了细致的指导，对我的科研能力进行了充分的锤炼，让我能够具备更好的独立思考和独立解决问题的能力。感谢王振华老师，是您在使用 Linux 系统时的信手拈来、运指如飞，让我知道了努

力的方向和自身的不足，让我有了进一步钻研 Linux 技术的动力。

百度七年，感谢百度运维部现任总监李硕、前任总监刘超两位对我的培养和指导，是你们让运维部形成了精钻技术、追求极致的良好氛围，带领大家成为国内一流的互联网运维团队。感谢我的第一任经理伏晔，带我进入了运维这一技术领域。感谢我的老经理苏阳对我在运维岗位的指导和支持。感谢臧志、王达心、曲显平对我在研发岗位、产品岗位上的指导和支持，让我能够不断挑战未知，勇敢前行。还有很多人需要感谢：我的历任经理、我的 VSOP 小伙伴们、我的 OPED 小伙伴们、我的 Image/Video 组小伙伴们，以及百度开放云组的小伙伴们。

古有云：学贵得师，亦贵得友。师也者，犹行路之有导也；友也者，犹陟险之有助也。请允许我用单独的段落向我的亦师亦友的老经理苏阳感恩，是你对我的悉心培养，让我更快地进入了职场角色，让我获得了加速的成长，让我真正成为了一名合格的运维工程师。指导和培养会谨记。

感谢我的导师马严教授、百度运维部总监李硕为本书做序，感谢滴滴联合创始人兼 CTO 张博的信任与支持，感谢张博、伏晔、臧志、窦喆、王达心、曲显平、陈江伟为本书撰写推荐语。谢谢你们的认可和鼓励！

最后，要感谢的是电子工业出版社的安娜，是你在人群中多看了我博客一眼，才让大家得以见到这本书的容颜。和安娜的沟通、合作过程，我感到非常的融洽和愉快，相信合作和友谊会一直延续。感谢设计师李玲的妙手丹青，让我对书的封面一见钟情。感谢电子工业出版社博文视点团队的领导和工作人员，是你们的专业态度和辛苦付出，才让这本书能够更好地呈现在读者面前。

最后的最后，感谢购买这本正版图书的读者们，你们对知识价值的认可和尊重，是所有技术作者努力和前进的源动力。